Part VII

Asset Pricing and Finance

CHAPTER

THIRTYTHREE

ASSET PRICING II: THE LUCAS ASSET PRICING MODEL

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install interpolation
```

33.1 Overview

As stated in an earlier lecture, an asset is a claim on a stream of prospective payments.

What is the correct price to pay for such a claim?

The elegant asset pricing model of Lucas [Luc78] attempts to answer this question in an equilibrium setting with risk-averse agents.

While we mentioned some consequences of Lucas' model earlier, it is now time to work through the model more carefully and try to understand where the fundamental asset pricing equation comes from.

A side benefit of studying Lucas' model is that it provides a beautiful illustration of model building in general and equilibrium pricing in competitive models in particular.

Another difference to our first asset pricing lecture is that the state space and shock will be continuous rather than discrete.

Let's start with some imports:

```
import numpy as np
from interpolation import interp
from numba import njit, prange
from scipy.stats import lognorm
import matplotlib.pyplot as plt
%matplotlib inline
```

33.2 The Lucas Model

Lucas studied a pure exchange economy with a representative consumer (or household), where

- *Pure exchange* means that all endowments are exogenous.
- *Representative* consumer means that either
  - there is a single consumer (sometimes also referred to as a household), or
  - all consumers have identical endowments and preferences

Either way, the assumption of a representative agent means that prices adjust to eradicate desires to trade.

This makes it very easy to compute competitive equilibrium prices.

33.2.1 Basic Setup

Let's review the setup.

Assets

There is a single "productive unit" that costlessly generates a sequence of consumption goods $\{y_t\}_{t=0}^{\infty}$.

Another way to view $\{y_t\}_{t=0}^{\infty}$ is as a *consumption endowment* for this economy.

We will assume that this endowment is Markovian, following the exogenous process

$$y_{t+1} = G(y_t, \xi_{t+1})$$

Here $\{\xi_t\}$ is an IID shock sequence with known distribution ϕ and $y_t \geq 0$.

An asset is a claim on all or part of this endowment stream.

The consumption goods $\{y_t\}_{t=0}^{\infty}$ are nonstorable, so holding assets is the only way to transfer wealth into the future.

For the purposes of intuition, it's common to think of the productive unit as a "tree" that produces fruit.

Based on this idea, a "Lucas tree" is a claim on the consumption endowment.

Consumers

A representative consumer ranks consumption streams $\{c_t\}$ according to the time separable utility functional

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t u(c_t) \tag{1}$$

Here

- $\beta \in (0, 1)$ is a fixed discount factor.
- u is a strictly increasing, strictly concave, continuously differentiable period utility function.
- $\mathbb{E}$ is a mathematical expectation.

33.2.2 Pricing a Lucas Tree

What is an appropriate price for a claim on the consumption endowment?

We'll price an *ex-dividend* claim, meaning that

- the seller retains this period's dividend
- the buyer pays p_t today to purchase a claim on
  - y_{t+1} and
  - the right to sell the claim tomorrow at price p_{t+1}

Since this is a competitive model, the first step is to pin down consumer behavior, taking prices as given.

Next, we'll impose equilibrium constraints and try to back out prices.

In the consumer problem, the consumer's control variable is the share π_t of the claim held in each period.

Thus, the consumer problem is to maximize (1) subject to

$$c_t + \pi_{t+1} p_t \leq \pi_t y_t + \pi_t p_t$$

along with $c_t \geq 0$ and $0 \leq \pi_t \leq 1$ at each t.

The decision to hold share π_t is actually made at time $t-1$.

But this value is inherited as a state variable at time t, which explains the choice of subscript.

The Dynamic Program

We can write the consumer problem as a dynamic programming problem.

Our first observation is that prices depend on current information, and current information is really just the endowment process up until the current period.

In fact, the endowment process is Markovian, so that the only relevant information is the current state $y \in \mathbb{R}_+$ (dropping the time subscript).

This leads us to guess an equilibrium where price is a function p of y.

Remarks on the solution method

- Since this is a competitive (read: price taking) model, the consumer will take this function p as given.
- In this way, we determine consumer behavior given p and then use equilibrium conditions to recover p.
- This is the standard way to solve competitive equilibrium models.

Using the assumption that price is a given function p of y, we write the value function and constraint as

$$v(\pi, y) = \max_{c, \pi'} \left\{ u(c) + \beta \int v(\pi', G(y, z)) \phi(dz) \right\}$$

subject to

$$c + \pi' p(y) \leq \pi y + \pi p(y) \tag{2}$$

We can invoke the fact that utility is increasing to claim equality in (2) and hence eliminate the constraint, obtaining

$$v(\pi, y) = \max_{\pi'} \left\{ u[\pi(y + p(y)) - \pi' p(y)] + \beta \int v(\pi', G(y, z)) \phi(dz) \right\} \tag{3}$$

The solution to this dynamic programming problem is an optimal policy expressing either π' or c as a function of the state (π, y).

- Each one determines the other, since $c(\pi, y) = \pi(y + p(y)) - \pi'(\pi, y)p(y)$

Next Steps

What we need to do now is determine equilibrium prices.

It seems that to obtain these, we will have to

1. Solve this two-dimensional dynamic programming problem for the optimal policy.
2. Impose equilibrium constraints.
3. Solve out for the price function $p(y)$ directly.

However, as Lucas showed, there is a related but more straightforward way to do this.

Equilibrium Constraints

Since the consumption good is not storable, in equilibrium we must have $c_t = y_t$ for all t.

In addition, since there is one representative consumer (alternatively, since all consumers are identical), there should be no trade in equilibrium.

In particular, the representative consumer owns the whole tree in every period, so $\pi_t = 1$ for all t.

Prices must adjust to satisfy these two constraints.

The Equilibrium Price Function

Now observe that the first-order condition for (3) can be written as

$$u'(c)p(y) = \beta \int v_1'(\pi', G(y, z))\phi(dz)$$

where v_1' is the derivative of v with respect to its first argument.

To obtain v_1' we can simply differentiate the right-hand side of (3) with respect to π, yielding

$$v_1'(\pi, y) = u'(c)(y + p(y))$$

Next, we impose the equilibrium constraints while combining the last two equations to get

$$p(y) = \beta \int \frac{u'[G(y, z)]}{u'(y)} [G(y, z) + p(G(y, z))]\phi(dz) \tag{4}$$

In sequential rather than functional notation, we can also write this as

$$p_t = \mathbb{E}_t \left[\beta \frac{u'(c_{t+1})}{u'(c_t)} (y_{t+1} + p_{t+1}) \right] \tag{5}$$

This is the famous consumption-based asset pricing equation.

Before discussing it further we want to solve out for prices.

33.2.3 Solving the Model

Equation (4) is a *functional equation* in the unknown function p.

The solution is an equilibrium price function p^*.

Let's look at how to obtain it.

Setting up the Problem

Instead of solving for it directly we'll follow Lucas' indirect approach, first setting

$$f(y) := u'(y)p(y) \tag{6}$$

so that (4) becomes

$$f(y) = h(y) + \beta \int f[G(y, z)]\phi(dz) \tag{7}$$

Here $h(y) := \beta \int u'[G(y, z)]G(y, z)\phi(dz)$ is a function that depends only on the primitives.

Equation (7) is a functional equation in f.

The plan is to solve out for f and convert back to p via (6).

To solve (7) we'll use a standard method: convert it to a fixed point problem.

First, we introduce the operator T mapping f into Tf as defined by

$$(Tf)(y) = h(y) + \beta \int f[G(y, z)]\phi(dz) \tag{8}$$

In what follows, we refer to T as the Lucas operator.

The reason we do this is that a solution to (7) now corresponds to a function f^* satisfying $(Tf^*)(y) = f^*(y)$ for all y.

In other words, a solution is a *fixed point* of T.

This means that we can use fixed point theory to obtain and compute the solution.

A Little Fixed Point Theory

Let $cb\mathbb{R}_+$ be the set of continuous bounded functions $f\colon \mathbb{R}_+ \to \mathbb{R}_+$.

We now show that

1. T has exactly one fixed point f^* in $cb\mathbb{R}_+$.
2. For any $f \in cb\mathbb{R}_+$, the sequence $T^k f$ converges uniformly to f^*.

(Note: If you find the mathematics heavy going you can take 1–2 as given and skip to the *next section*)

Recall the Banach contraction mapping theorem.

It tells us that the previous statements will be true if we can find an $\alpha < 1$ such that

$$\|Tf - Tg\| \leq \alpha \|f - g\|, \qquad \forall\, f, g \in cb\mathbb{R}_+ \tag{9}$$

Here $\|h\| := \sup_{x \in \mathbb{R}_+} |h(x)|$.

To see that (9) is valid, pick any $f, g \in cb\mathbb{R}_+$ and any $y \in \mathbb{R}_+$.

Observe that, since integrals get larger when absolute values are moved to the inside,

$$
\begin{aligned}
|Tf(y) - Tg(y)| &= \left| \beta \int f[G(y,z)]\phi(dz) - \beta \int g[G(y,z)]\phi(dz) \right| \\
&\leq \beta \int |f[G(y,z)] - g[G(y,z)]| \, \phi(dz) \\
&\leq \beta \int \|f - g\| \phi(dz) \\
&= \beta \|f - g\|
\end{aligned}
$$

Since the right-hand side is an upper bound, taking the sup over all y on the left-hand side gives (9) with $\alpha := \beta$.

33.2.4 Computation – An Example

The preceding discussion tells that we can compute f^* by picking any arbitrary $f \in cb\mathbb{R}_+$ and then iterating with T.

The equilibrium price function p^* can then be recovered by $p^*(y) = f^*(y)/u'(y)$.

Let's try this when $\ln y_{t+1} = \alpha \ln y_t + \sigma \epsilon_{t+1}$ where $\{\epsilon_t\}$ is IID and standard normal.

Utility will take the isoelastic form $u(c) = c^{1-\gamma}/(1-\gamma)$, where $\gamma > 0$ is the coefficient of relative risk aversion.

We will set up a `LucasTree` class to hold parameters of the model

```python
class LucasTree:
    """
    Class to store parameters of the Lucas tree model.

    """

    def __init__(self,
                 γ=2,            # CRRA utility parameter
                 β=0.95,         # Discount factor
                 α=0.90,         # Correlation coefficient
                 σ=0.1,          # Volatility coefficient
                 grid_size=100):

        self.γ, self.β, self.α, self.σ = γ, β, α, σ

        # Set the grid interval to contain most of the mass of the
        # stationary distribution of the consumption endowment
        ssd = self.σ / np.sqrt(1 - self.α**2)
        grid_min, grid_max = np.exp(-4 * ssd), np.exp(4 * ssd)
        self.grid = np.linspace(grid_min, grid_max, grid_size)
        self.grid_size = grid_size

        # Set up distribution for shocks
        self.ϕ = lognorm(σ)
        self.draws = self.ϕ.rvs(500)

        self.h = np.empty(self.grid_size)
        for i, y in enumerate(self.grid):
            self.h[i] = β * np.mean((y**α * self.draws)**(1 - γ))
```

The following function takes an instance of the `LucasTree` and generates a jitted version of the Lucas operator

```python
def operator_factory(tree, parallel_flag=True):

    """
    Returns approximate Lucas operator, which computes and returns the
    updated function Tf on the grid points.

    tree is an instance of the LucasTree class

    """

    grid, h = tree.grid, tree.h
    α, β = tree.α, tree.β
    z_vec = tree.draws

    @njit(parallel=parallel_flag)
    def T(f):
        """
        The Lucas operator
        """

        # Turn f into a function
        Af = lambda x: interp(grid, f, x)

        Tf = np.empty_like(f)
        # Apply the T operator to f using Monte Carlo integration
        for i in prange(len(grid)):
            y = grid[i]
            Tf[i] = h[i] + β * np.mean(Af(y**α * z_vec))

        return Tf

    return T
```

To solve the model, we write a function that iterates using the Lucas operator to find the fixed point.

```python
def solve_model(tree, tol=1e-6, max_iter=500):
    """
    Compute the equilibrium price function associated with Lucas
    tree

    * tree is an instance of LucasTree

    """
    # Simplify notation
    grid, grid_size = tree.grid, tree.grid_size
    γ = tree.γ

    T = operator_factory(tree)

    i = 0
    f = np.ones_like(grid)  # Initial guess of f
    error = tol + 1
    while error > tol and i < max_iter:
        Tf = T(f)
        error = np.max(np.abs(Tf - f))
        f = Tf
        i += 1
```

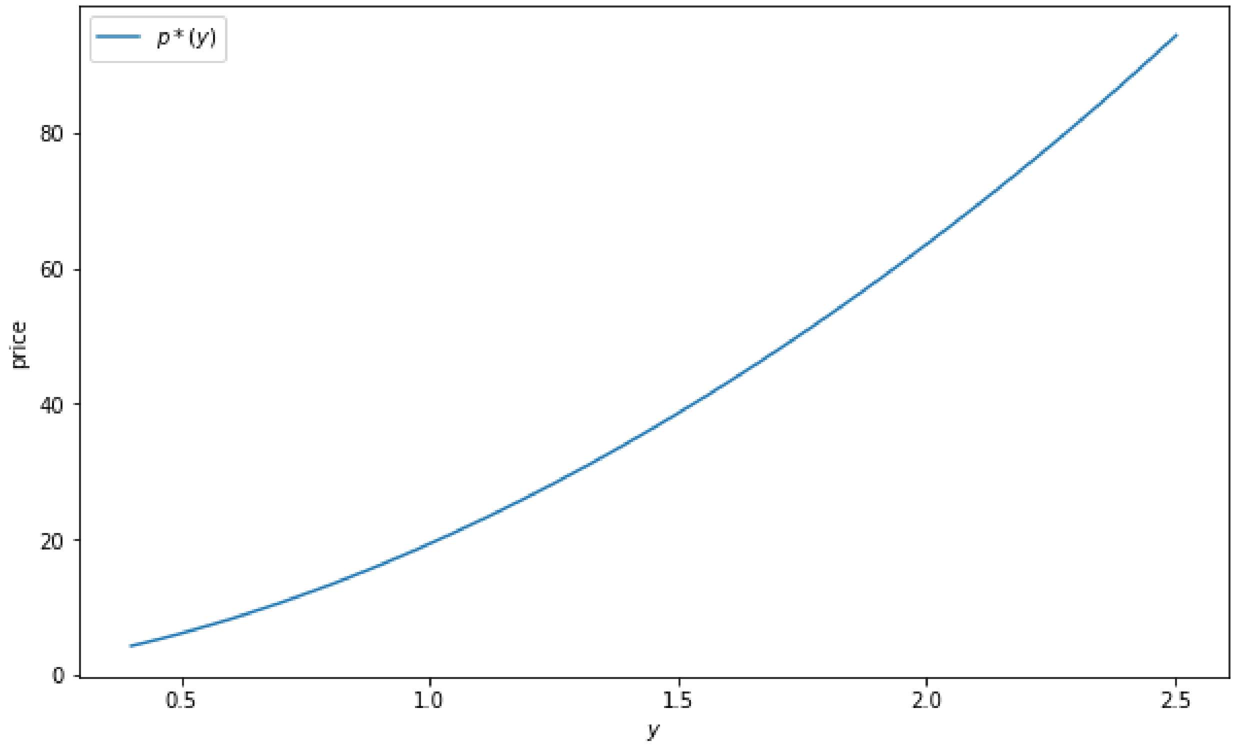
(continues on next page)

(continued from previous page)

```
    price = f * grid**y  # Back out price vector

    return price
```

Solving the model and plotting the resulting price function

```
tree = LucasTree()
price_vals = solve_model(tree)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(tree.grid, price_vals, label='$p*(y)$')
ax.set_xlabel('$y$')
ax.set_ylabel('price')
ax.legend()
plt.show()
```



We see that the price is increasing, even if we remove all serial correlation from the endowment process.

The reason is that a larger current endowment reduces current marginal utility.

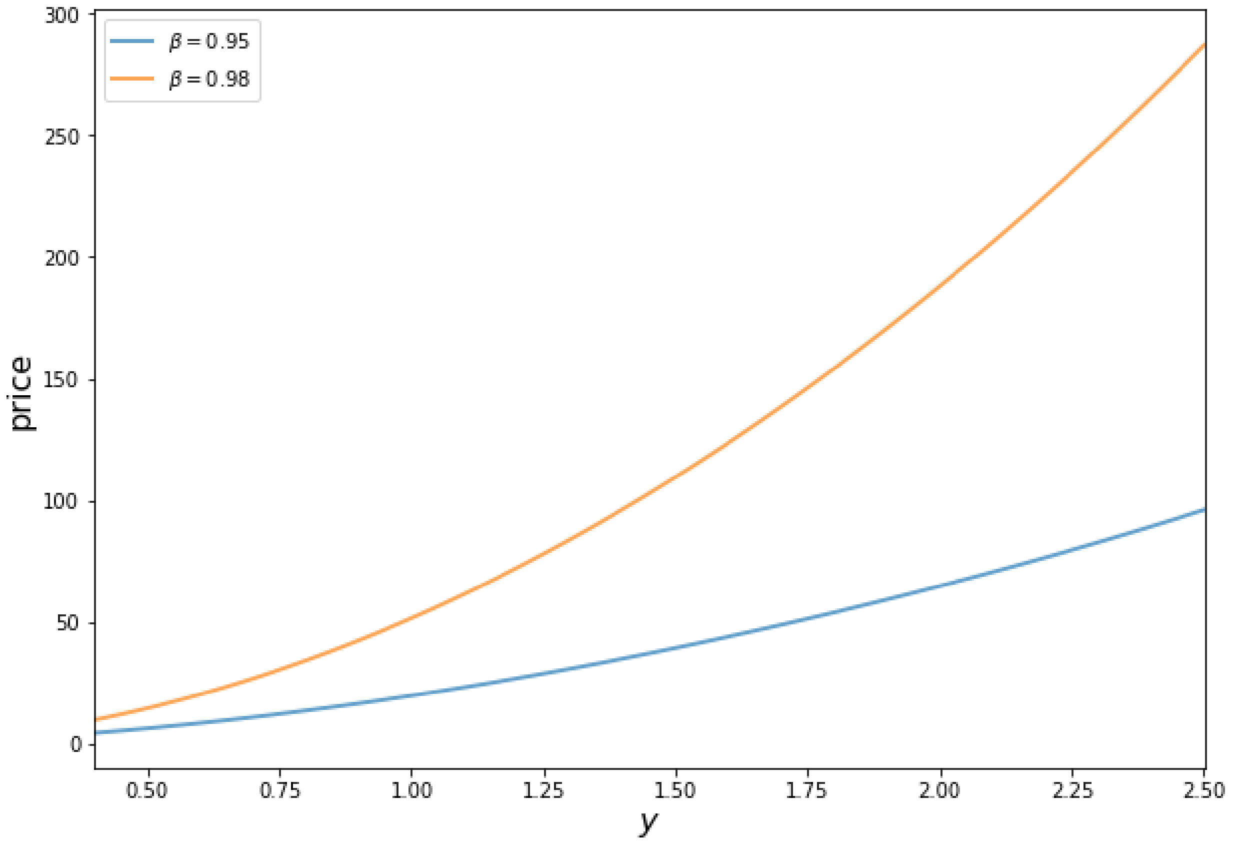
The price must therefore rise to induce the household to consume the entire endowment (and hence satisfy the resource constraint).

What happens with a more patient consumer?

Here the orange line corresponds to the previous parameters and the green line is price when $\beta = 0.98$.

We see that when consumers are more patient the asset becomes more valuable, and the price of the Lucas tree shifts up.

Exercise 1 asks you to replicate this figure.


$\beta = 0.95$
$\beta = 0.98$
price
y
300
250
200
150
100
50
0
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50

33.3 Exercises

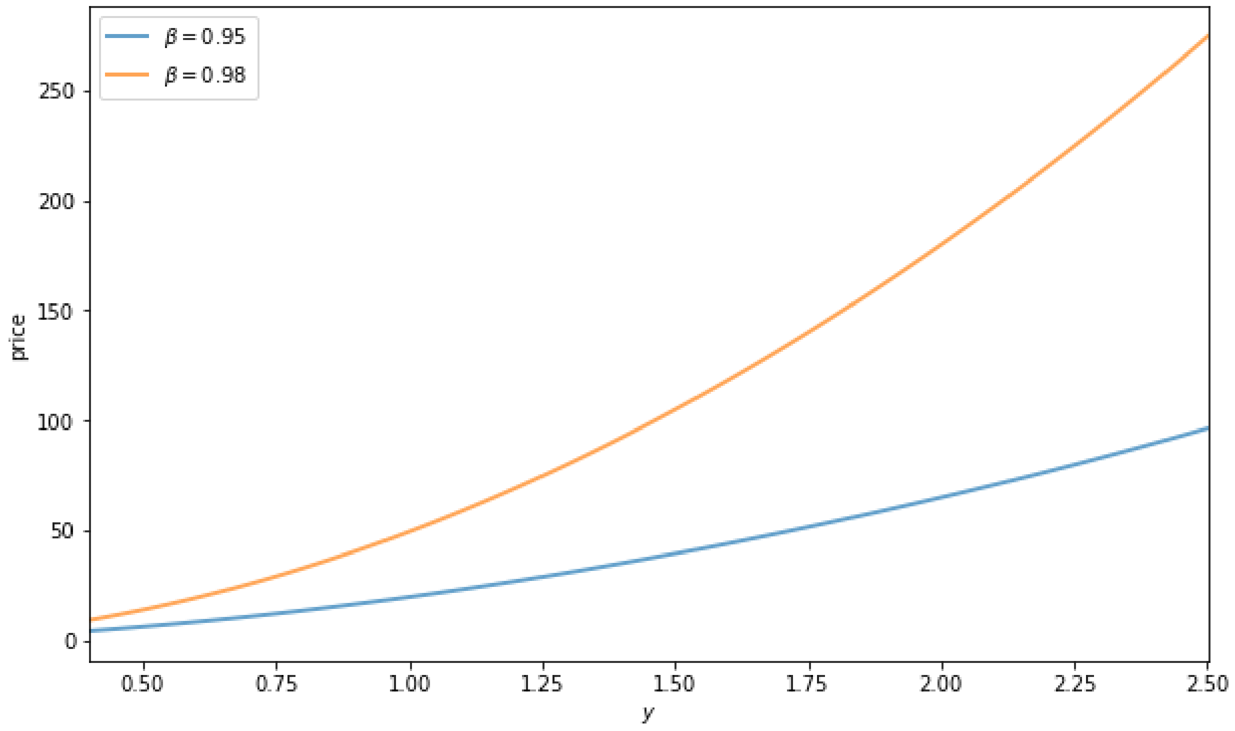
33.3.1 Exercise 1

Replicate *the figure* to show how discount factors affect prices.

33.4 Solutions

33.4.1 Exercise 1

```
fig, ax = plt.subplots(figsize=(10, 6))

for β in (.95, 0.98):
    tree = LucasTree(β=β)
    grid = tree.grid
    price_vals = solve_model(tree)
    label = rf'$\beta = {β}$'
    ax.plot(grid, price_vals, lw=2, alpha=0.7, label=label)

ax.legend(loc='upper left')
ax.set(xlabel='$y$', ylabel='price', xlim=(min(grid), max(grid)))
plt.show()
```



CHAPTER THIRTYFOUR

ELEMENTARY ASSET PRICING THEORY

Contents

34.1 Overview

This lecture summarizes the heart of applied asset-pricing theory.

From a single equation, we'll derive

- a mean-variance frontier
- a single-factor model of excess returns on each member of a collection of assets

To do this, we use two ideas:

- an asset pricing equation
- a Cauchy-Schwartz inequality

For background and basic concepts, see our lecture orthogonal projections and their applications.

As a sequel to the material here, please see our lecture two modifications of mean-variance portfolio theory.

34.2 Key Equation

We begin with a **key asset pricing equation**:

$$EmR^i = 1 \tag{1}$$

for $i = 1, \ldots, I$ and where

$$\begin{aligned} m &= \text{ stochastic discount factor} \\ R^i &= \text{random gross return on asset } i \\ E &\sim \text{ mathematical expectation} \end{aligned}$$

The random gross returns R^i and the scalar stochastic discount factor m live live in a common probability space.

[HR87] and [HJ91] explain how the existence of a scalar stochastic discount factor that verifies equation (1) is implied by a **law of one price** that requires that all portfolios of assets that end up having the same payouts must have the same price.

They also explain how the **absence of an arbitrage** implies that the stochastic discount factor $m \geq 0$.

34.3 Implications of Key Equation

We combine key equation (1) with a remark of Lars Peter Hansen that "asset pricing theory is all about covariances".

Note: Lars Hansen's remark is a concise summary of ideas in [HR87] and [HJ91]. For other important foundations of these ideas, see [Ros76], [Ros78], [HK79], [Kre81], and [CR83].

By that remark, Lars Hansen meant that interesting restrictions can be deduced by recognizing that EmR^i is a component of the covariance between m and R^i and then using that fact to rearrange key equation (1).

Let's do this step by step.

First note that the definition $\operatorname{cov}(m, R^i) = E(m - Em)(R^i - ER^i)$ of a covariance implies that

$$EmR^i = EmER^i + \operatorname{cov}(m, R^i)$$

Substituting this result into key equation (1) gives

$$1 = EmER^i + \operatorname{cov}(m, R^i) \tag{2}$$

Next note that for a risk-free asset with non-random gross return R^f, equation (1) becomes

$$1 = ER^f m = R^f Em.$$

This is true because we can pull the constant R^f outside the mathematical expectation.

It follows that the gross return on a risk-free asset is

$$R^f = 1/E(m)$$

Using this formula for R^f in equation (2) and rearranging, it follows that

$$R^f = ER^i + \operatorname{cov}(m, R^i) R^f$$

which can be rearranged to become

$$ER^i = R^f - \text{cov}\left(m, R^i\right) R^f.$$

It follows that we can express an **excess return** $ER^i - R^f$ on asset i relative to the risk-free rate as

$$ER^i - R^f = -\text{cov}\left(m, R^i\right) R^f \tag{3}$$

Equation (3) can be rearranged to display important parts of asset pricing theory.

34.4 Expected Return - Beta Representation

We can obtain the celebrated **expected-return-Beta -representation** for gross return R^i simply by rearranging excess return equation (3) to become

$$ER^i = R^f + \left(\underbrace{\frac{\text{cov}\left(R^i, m\right)}{\text{var}(m)}}_{\beta_{i,m}=\text{regression coefficient}}\right)\left(\underbrace{-\frac{\text{var}(m)}{E(m)}}_{\lambda_m=\text{price of risk}}\right)$$

or

$$ER^i = R^f + \beta_{i,m}\lambda_m \tag{4}$$

Here

- $\beta_{i,m}$ is a (population) least squares regression coefficient of gross return R^i on stochastic discount factor m, an object that is often called asset i's **beta**
- λ_m is minus the variance of m divided by the mean of m, an object that is often called the **price of risk**.

To interpret this representation it helps to provide the following widely used example.

Example

A popular model of m is

$$m_{t+1} = \exp(-\rho)\exp(-\gamma(c_{t+1} - c_t))$$

where $\rho > 0$, $\gamma > 0$, and the log of consumption growth is governed by

$$c_{t+1} - c_t = \mu + \sigma_c \epsilon_{t+1}$$

where $\epsilon_{t+1} \sim \mathcal{N}(0, 1)$.

Here

- $\gamma > 0$ is a coefficient of relative risk aversion
- $\rho > 0$ is a fixed intertemporal discount rate

$$m_{t+1} = \exp(-\rho)\exp(-\gamma\mu - \gamma\sigma_c\epsilon_{t+1})$$

In this case

$$Em_{t+1} = \exp(-\rho)\exp\left(-\gamma\mu + \frac{\sigma_c^2\gamma^2}{2}\right)$$

and

$$\text{var}(m_{t+1}) = E(m)[\exp(\sigma_c^2\gamma^2) - 1)]$$

When $\gamma > 0$, it is true that

- when consumption growth is **high**, m is **low**
- when consumption growth is **low**, m is **high**

According the representation (4), an asset with an R^i that can be expected to be high when consumption growth is low has β_i positive and a low expected return.

- because it has a high gross return when consumption growth is low, it is a good hedge against consumption risk, which justifies its low average return

An asset with an R^i that is low when consumption growth is low has β_i negative and a high expected return.

- because it has a low gross return when consumption growth is low, it is a poor hedge against consumption risk, which justifies its high average return

34.5 Mean-Variance Frontier

Now we'll derive the celebrated **mean-variance frontier**.

We do this using a classic method of Lars Peter Hansen and Scott Richard [HR87].

Note: Methods of Hansen and Richard are described and used extensively by [Coc05].

Their idea was rearrange the key equation (1), namely, $EmR^i = 1$, and then to apply the Cauchy-Schwarz inequality.

A convenient way to remember the Cauchy-Schwartz inequality in our context is that it says that an R^2 in any regression has to be less than or equal to 1.

Let's apply that idea to deduce

$$1 = E\left(mR^i\right) = E(m)E\left(R^i\right) + \rho_{m,R^i}\frac{\sigma(m)}{E(m)}\sigma\left(R^i\right) \tag{5}$$

where ρ_{m,R^i} is the correlation coefficient defined as

$$\rho_{m,R^i} \equiv \frac{\text{cov}\left(m, R^i\right)}{\sigma(m)\sigma\left(R^i\right)}$$

and where σ denotes the standard deviation of the variable in parentheses

Equation (5) implies

$$ER^i = R^f - \rho_{m,R^i}\frac{\sigma(m)}{E(m)}\sigma\left(R^i\right)$$

Because $\rho_{m,R^i} \in [-1, 1]$, it follows that $|\rho_{m,R^i}| \leq 1$ and that

$$\left|ER^i - R^f\right| \leqslant \frac{\sigma(m)}{E(m)}\sigma\left(R^i\right) \tag{6}$$

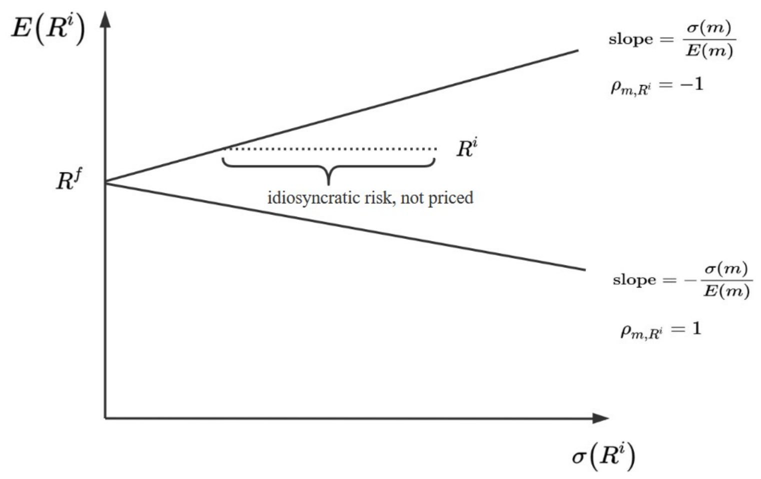
Inequality (6) delineates a **mean-variance frontier**

(Actually, it looks more like a **mean-standard-deviation frontier**)

Evidently, points on the frontier correspond to gross returns that are perfectly correlated (either positively or negatively) with the stochastic discount factor m.

We summarize this observation as

$$\rho_{m,R^i} = \begin{cases} +1 & \implies R^i \text{ is on lower frontier} \\ -1 & \implies R^i \text{ is on an upper frontier} \end{cases}$$



$E(R^i)$
slope $= \frac{\sigma(m)}{E(m)}$
$\rho_{m,R^i} = -1$
R^i
R^f
idiosyncratic risk, not priced
slope $= -\frac{\sigma(m)}{E(m)}$
$\rho_{m,R^i} = 1$
$\sigma(R^i)$

The image below illustrates a mean-variance frontier.

The mathematical structure of the mean-variance frontier described by inequality (6) implies that

- all returns on frontier are perfectly correlated.

  Thus,

  – Let R^m, R^{mv} be two returns on frontier.

  – Then for some scalar a

  – $R^{mv} = R^f + a\left(R^m - R^f\right)$

  This is an **exact** equation with no **residual**

- each return R^i that is on the mean-variance frontier is perfectly correlated with m

  – $\left(\rho_{m,R^i} = -1\right) \Rightarrow \begin{cases} m = a + bR^{mv} \\ R^{mv} = e + dm \end{cases}$ for some scalars a, b, e, d,

  Therefore, **any return on the mean-variance frontier is a legitimate stochastic discount factor**

- for any mean-variance-efficient return R^{mv} that is on the frontier but that is **not** R^f, there exists a **single-beta representation** for any return R^i that takes the form:

$$ER^i = R^f + \beta_{i,R^{mv}}\left[E\left(R^{mv}\right) - R^f\right] \tag{7}$$

- The special case of a single-beta representation (7) with $R^i = R^{mv}$ is

  $ER^{mv} = R^f + 1 \cdot \left[E\left(R^{mv}\right) - R^f\right]$

34.6 Empirical Implementations

We briefly describe empirical implementations of multi-factor generalizations of the single-factor model described above.

The single-beta representation (7) is a special case with there being just a single factor.

Two representations are often used in empirical work.

One is a **time-series regression** of gross return R_t^i on multiple risk factors $f_t^j, j = a, b, \ldots$ that is designed to uncover exposures of return R^i to each of a set of **risk-factors** $f_t^j, j = a, b, \ldots,$:

$$\begin{aligned} R_t^i = a_i + \beta_{i,a} f_t^a + \beta_{i,b} f_t^b + \ldots + \epsilon_t^i, \quad t = 1, 2, \ldots, T \\ \epsilon_t^i \perp f_t^j, i = 1, 2, \ldots, I; j = a, b, \ldots \end{aligned}$$

For example:

- a popular **single-factor** model specifies the single factor f_t to be the return on the market portfolio
- another popular **single-factor** model called the consumption based model specifies the factor to be $m_{t+1} = \beta \frac{u'(c_{t+1})}{u'(c_t)}$, where c_t is a representative consumer's time t consumption.

Model objects are interpreted as follows:

- $\beta_{i,a}$ is the exposure of return R^i to factor f_a risk
- λ_a is the price of exposure to factor f_a risk

The other representation entails a **cross-section regression** of **average returns** ER^i for assets $i = 1, 2, \ldots, I$ on **prices of risk** λ_j for $j = a, b, c, \ldots$

Here is the regression specification:

$$ER^i = \gamma + \beta_{i,a}\lambda_a + \beta_{i,b}\lambda_b + \cdots$$

Testing strategies:

Time-series and cross-section regressions play roles in both **estimating** and **testing** beta representation models.

The basic idea is to implement the following two steps.

Step 1:

- Estimate $a_i, \beta_{i,a}, \beta_{i,b}, \cdots$ by running a **time series regression:** R_t^i on a constant and $f_t^a, f_t^b, \ldots$

Step 2:

- take the $\beta_{i,j}$'s estimated in step one as regressors together with data on average returns ER^i over some period and then estimate the **cross-section regression**

$$\underbrace{E\left(R^i\right)}_{\text{average return over time series}} = \gamma + \underbrace{\beta_{i,a}}_{\text{regressor}} \underbrace{\lambda_a}_{\text{regressioncoefficient}} + \underbrace{\beta_{i,b}}_{\text{regressor}} \underbrace{\lambda_b}_{\text{regressioncoefficient}} + \cdots + \underbrace{\alpha_i}_{\text{pricing errors}}, i = 1, \ldots, I; \quad \underbrace{\alpha_i \perp \beta_{i,j}, j = a, b, \ldots}_{\text{least squares orthogonality conditio}}$$

- estimate $\gamma, \lambda_a, \lambda_b, \ldots$ by an appropriate regression technique, being thoughtful about recognizing that the regressors have been generated by a step 1 regression.

Note that presumably the risk-free return $ER^f = \gamma$.

For excess returns $R^{ei} = R^i - R^f$ we have

$$ER^{ei} = \beta_{i,a}\lambda_a + \beta_{i,b}\lambda_b + \cdots + \alpha_i, i = 1, \ldots, I$$

34.7 Exercises

Let's start with some imports.

```
import numpy as np
from scipy.stats import stats
import statsmodels.api as sm
from statsmodels.sandbox.regression.gmm import GMM
import matplotlib.pyplot as plt
%matplotlib inline
```

Lots of our calculations will involve computing population and sample OLS regressions.

So we define a function for simple univariate OLS regression that calls the `OLS` routine from `statsmodels`.

```
def simple_ols(X, Y, constant=False):

    if constant:
        X = sm.add_constant(X)

    model = sm.OLS(Y, X)
    res = model.fit()

    β_hat = res.params[-1]
    σ_hat = np.sqrt(res.resid @ res.resid / res.df_resid)

    return β_hat, σ_hat
```

34.7.1 Exercise 1

Look at the equation,

$$R^i_t - R^f = \beta_{i,R^m}(R^m_t - R^f) + \sigma_i \varepsilon_{i,t}.$$

Verify that this equation is a regression equation.

34.7.2 Exercise 2

Give a formula for the regression coefficient β_{i,R^m}.

34.7.3 Exercise 3

Recall our earlier discussions of a **direct problem** and an **inverse problem**.

- A direct problem is about simulating a particular model.
- An inverse problem is about using data to **estimate** or **choose** a particular model from a manifold of models.

Please assume the parameter values set below and then simulate 2000 observations from the theory specified above for 5 assets, $i = 1, \ldots, 5$.

\begin{align*} E\left[R^f\right] &= 0.02 \ \sigma_f &= 0.00 \ \xi &= 0.06 \ \lambda &= 0.04 \ \beta_{1, R^m} &= 0.2 \ \sigma_1 &= 0.04 \ \beta_{2, R^m} &= .4 \ \sigma_2 &= 0.04 \ \beta_{3, R^m} &= .6 \ \sigma_3 &= 0.04 \ \beta_{4, R^m} &= .8 \ \sigma_4 &= 0.04 \ \beta_{5, R^m} &= 1.0 \ \sigma_5 &= 0.04 \end{align*}

More Exercises

Now come some even more fun parts!

Our theory implies that there exist values of two scalars, a and b, such that a legitimate stochastic discount factor is:

$$m_t = a + bR_t^m$$

The parameters a, b must satisfy the following equations:

\begin{align*} E[(a + b R_t^m) R^m_t)] &= 1 \ E[(a + b R_t^m) R^f_t)] &= 1 \end{align*}

34.7.4 Exercise 4

Using the equations above, find a system of two **linear** equations that you can solve for a and b as functions of the parameters $(\lambda, \xi, E[R_f])$.

Write a function that can solve these equations.

Please check the **condition number** of a key matrix that must be inverted to determine a, b

34.7.5 Exercise 5

Using the estimates of the parameters that you generated above, compute the implied stochastic discount factor.

34.8 Solutions

34.8.1 Solution to Exercise 1

To verify that it is a **regression equation** we must show that the residual is orthogonal to the regressor.

Our assumptions about mutual orthogonality imply that

$$E\left[\epsilon_{i,t}\right] = 0, \quad E\left[\epsilon_{i,t} u_t\right] = 0$$

It follows that

$$\begin{aligned} E\left[\sigma_i \epsilon_{i,t}\left(R_t^m - R^f\right)\right] &= E\left[\sigma_i \epsilon_{i,t}\left(\xi + \lambda u_t\right)\right] \\ &= \sigma_i \xi E\left[\epsilon_{i,t}\right] + \sigma_i \lambda E\left[\epsilon_{i,t} u_t\right] \\ &= 0 \end{aligned}$$

34.8.2 Solution to Exercise 2

The regression coefficient β_{i,R^m} is

$$\beta_{i,R^m} = \frac{Cov\left(R_t^i - R^f, R_t^m - R^f\right)}{Var\left(R_t^m - R^f\right)}$$

34.8.3 Solution to Exercise 3

Direct Problem:

```
# Code for the direct problem

# assign the parameter values
ERf = 0.02
σf = 0.00 # Zejin: Hi tom, here is where you manipulate σf
ξ = 0.06
λ = 0.08
βi = np.array([0.2, .4, .6, .8, 1.0])
σi = np.array([0.04, 0.04, 0.04, 0.04, 0.04])
```

```
# in this cell we set the number of assets and number of observations
# we first set T to a large number to verify our computation results
T = 2000
N = 5
```

```
# simulate i.i.d. random shocks
e = np.random.normal(size=T)
u = np.random.normal(size=T)
ϵ = np.random.normal(size=(N, T))
```

```
# simulate the return on a risk-free asset
Rf = ERf + σf * e

# simulate the return on the market portfolio
excess_Rm = ξ + λ * u
Rm = Rf + excess_Rm

# simulate the return on asset i
Ri = np.empty((N, T))
for i in range(N):
    Ri[i, :] = Rf + βi[i] * excess_Rm + σi[i] * ϵ[i, :]
```

Now that we have a panel of data, we'd like to solve the inverse problem by assuming the theory specified above and estimating the coefficients given above.

```
# Code for the inverse problem
```

Inverse Problem:

We will solve the inverse problem by simple OLS regressions.

1. estimate $E[R^f]$ and σ_f

```
ERf_hat, σf_hat = simple_ols(np.ones(T), Rf)
```

```
ERf_hat, σf_hat
```

```
(0.020000000000000003, 3.123283175179055e-17)
```

Let's compare these with the *true* population parameter values.

```
ERf, σf
```

```
(0.02, 0.0)
```

1. ξ and λ

```
ξ_hat, λ_hat = simple_ols(np.ones(T), Rm - Rf)
```

```
ξ_hat, λ_hat
```

```
(0.06155774900629886, 0.07764035983032216)
```

```
ξ, λ
```

```
(0.06, 0.08)
```

1. β_{i,R^m} and σ_i

```
βi_hat = np.empty(N)
σi_hat = np.empty(N)

for i in range(N):
    βi_hat[i], σi_hat[i] = simple_ols(Rm - Rf, Ri[i, :] - Rf)
```

```
βi_hat, σi_hat
```

```
(array([0.20099261, 0.40911167, 0.58436921, 0.79611864, 0.98288084]),
 array([0.04144067, 0.04021236, 0.04049461, 0.04020027, 0.03947831]))
```

```
βi, σi
```

```
(array([0.2, 0.4, 0.6, 0.8, 1. ]), array([0.04, 0.04, 0.04, 0.04, 0.04]))
```

Q: How close did your estimates come to the parameters we specified?

34.8.4 Solution to Exercise 4

\begin{align} a ((E(R^f) + \xi) + b ((E(R^f) + \xi)^2 + \lambda^2 + \sigma_f^2) & =1 \cr a E(R^f) + b (E(R^f)^2 + \xi E(R^f) + \sigma_f ^ 2) & = 1 \end{align}

```
# Code here
def solve_ab(ERf, σf, λ, ξ):

    M = np.empty((2, 2))
    M[0, 0] = ERf + ξ
    M[0, 1] = (ERf + ξ) ** 2 + λ ** 2 + σf ** 2
    M[1, 0] = ERf
    M[1, 1] = ERf ** 2 + ξ * ERf + σf ** 2

    a, b = np.linalg.solve(M, np.ones(2))
    condM = np.linalg.cond(M)

    return a, b, condM
```

Let's try to solve a and b using the actual model parameters.

```
a, b, condM = solve_ab(ERf, σf, λ, ξ)
```

```
a, b, condM
```

```
(87.49999999999999, -468.7499999999999, 54.406619883717504)
```

34.8.5 Solution to Exercise 5

Now let's pass $\hat{E}(R^f), \hat{\sigma}^f, \hat{\lambda}, \hat{\xi}$ to the function `solve_ab`.

```
a_hat, b_hat, M_hat = solve_ab(ERf_hat, σf_hat, λ_hat, ξ_hat)
```

```
a_hat, b_hat, M_hat
```

```
(91.64308415764174, -510.59629115592224, 59.82959629021462)
```

CHAPTER

THIRTYFIVE

TWO MODIFICATIONS OF MEAN-VARIANCE PORTFOLIO THEORY

Contents

35.1 Overview

35.1.1 Remarks About Estimating Means and Variances

The famous **Black-Litterman** (1992) [BL92] portfolio choice model that we describe in this lecture is motivated by the finding that with high or moderate frequency data, means are more difficult to estimate than variances.

A model of **robust portfolio choice** that we'll describe also begins from the same starting point.

To begin, we'll take for granted that means are more difficult to estimate that covariances and will focus on how Black and Litterman, on the one hand, an robust control theorists, on the other, would recommend modifying the **mean-variance portfolio choice model** to take that into account.

At the end of this lecture, we shall use some rates of convergence results and some simulations to verify how means are more difficult to estimate than variances.

Among the ideas in play in this lecture will be

- Mean-variance portfolio theory
- Bayesian approaches to estimating linear regressions
- A risk-sensitivity operator and its connection to robust control theory

Let's start with some imports:

```
import numpy as np
import scipy as sp
import scipy.stats as stat
import matplotlib.pyplot as plt
%matplotlib inline
from ipywidgets import interact, FloatSlider
```

35.1.2 Adjusting Mean-variance Portfolio Choice Theory for Distrust of Mean Excess Returns

This lecture describes two lines of thought that modify the classic mean-variance portfolio choice model in ways designed to make its recommendations more plausible.

As we mentioned above, the two approaches build on a common and widespread hunch – that because it is much easier statistically to estimate covariances of excess returns than it is to estimate their means, it makes sense to contemplated the consequences of adjusting investors' subjective beliefs about mean returns in order to render more sensible decisions.

Both of the adjustments that we describe are designed to confront a widely recognized embarrassment to mean-variance portfolio theory, namely, that it usually implies taking very extreme long-short portfolio positions.

35.1.3 Mean-variance Portfolio Choice

A risk-free security earns one-period net return r_f.

An $n \times 1$ vector of risky securities earns an $n \times 1$ vector $\vec{r} - r_f \mathbf{1}$ of *excess returns*, where $\mathbf{1}$ is an $n \times 1$ vector of ones.

The excess return vector is multivariate normal with mean μ and covariance matrix Σ, which we express either as

$$\vec{r} - r_f \mathbf{1} \sim \mathcal{N}(\mu, \Sigma)$$

or

$$\vec{r} - r_f \mathbf{1} = \mu + C\epsilon$$

where $\epsilon \sim \mathcal{N}(0, I)$ is an $n \times 1$ random vector.

Let w be an $n \times 1$ vector of portfolio weights.

A portfolio consisting w earns returns

$$w'(\vec{r} - r_f \mathbf{1}) \sim \mathcal{N}(w'\mu, w'\Sigma w)$$

The **mean-variance portfolio choice problem** is to choose w to maximize

$$U(\mu, \Sigma; w) = w'\mu - \frac{\delta}{2} w'\Sigma w \tag{1}$$

where $\delta > 0$ is a risk-aversion parameter. The first-order condition for maximizing (1) with respect to the vector w is

$$\mu = \delta \Sigma w$$

which implies the following design of a risky portfolio:

$$w = (\delta \Sigma)^{-1} \mu \tag{2}$$

35.1.4 Estimating the Mean and Variance

The key inputs into the portfolio choice model (2) are

- estimates of the parameters μ, Σ of the random excess return vector$(\vec{r} - r_f \mathbf{1})$
- the risk-aversion parameter δ

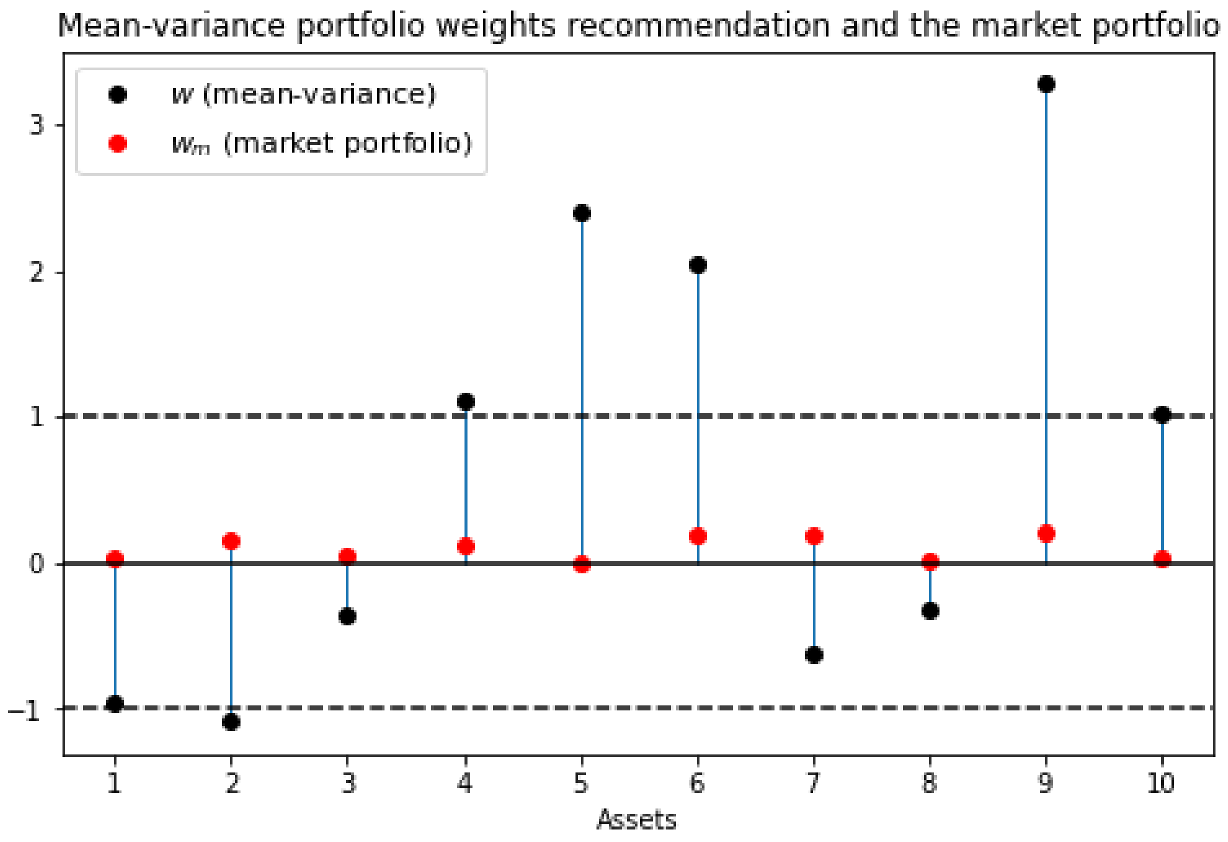
A standard way of estimating μ is maximum-likelihood or least squares; that amounts to estimating μ by a sample mean of excess returns and estimating Σ by a sample covariance matrix.

35.1.5 The Black-Litterman Starting Point

When estimates of μ and Σ from historical sample means and covariances have been combined with **reasonable** values of the risk-aversion parameter δ to compute an optimal portfolio from formula (2), a typical outcome has been $w$'s with **extreme long and short positions**.

A common reaction to these outcomes is that they are so unreasonable that a portfolio manager cannot recommend them to a customer.

```python
np.random.seed(12)

N = 10                                                  # Number of assets
T = 200                                                 # Sample size

# random market portfolio (sum is normalized to 1)
w_m = np.random.rand(N)
w_m = w_m / (w_m.sum())

# True risk premia and variance of excess return (constructed
# so that the Sharpe ratio is 1)
μ = (np.random.randn(N) + 5)  /100      # Mean excess return (risk premium)
S = np.random.randn(N, N)        # Random matrix for the covariance matrix
V = S @ S.T           # Turn the random matrix into symmetric psd
# Make sure that the Sharpe ratio is one
Σ = V * (w_m @ μ)**2 / (w_m @ V @ w_m)

# Risk aversion of market portfolio holder
δ = 1 / np.sqrt(w_m @ Σ @ w_m)

# Generate a sample of excess returns
excess_return = stat.multivariate_normal(μ, Σ)
sample = excess_return.rvs(T)

# Estimate μ and Σ
μ_est = sample.mean(0).reshape(N, 1)
Σ_est = np.cov(sample.T)

w = np.linalg.solve(δ * Σ_est, μ_est)

fig, ax = plt.subplots(figsize=(8, 5))
ax.set_title('Mean-variance portfolio weights recommendation and the market portfolio
 ↪')
ax.plot(np.arange(N)+1, w, 'o', c='k', label='$w$ (mean-variance)')
ax.plot(np.arange(N)+1, w_m, 'o', c='r', label='$w_m$ (market portfolio)')
ax.vlines(np.arange(N)+1, 0, w, lw=1)
ax.vlines(np.arange(N)+1, 0, w_m, lw=1)
ax.axhline(0, c='k')
ax.axhline(-1, c='k', ls='--')
ax.axhline(1, c='k', ls='--')
ax.set_xlabel('Assets')
ax.xaxis.set_ticks(np.arange(1, N+1, 1))
plt.legend(numpoints=1, fontsize=11)
plt.show()
```



Black and Litterman's responded to this situation in the following way:

- They continue to accept (2) as a good model for choosing an optimal portfolio w.
- They want to continue to allow the customer to express his or her risk tolerance by setting δ.
- Leaving Σ at its maximum-likelihood value, they push μ away from its maximum-likelihood value in a way designed to make portfolio choices that are more plausible in terms of conforming to what most people actually do.

In particular, given Σ and a reasonable value of δ, Black and Litterman reverse engineered a vector μ_{BL} of mean excess returns that makes the w implied by formula (2) equal the **actual** market portfolio w_m, so that

$$w_m = (\delta\Sigma)^{-1}\mu_{BL}$$

35.1.6 Details

Let's define

$$w_m'\mu \equiv (r_m - r_f)$$

as the (scalar) excess return on the market portfolio w_m.

Define

$$\sigma^2 = w_m'\Sigma w_m$$

as the variance of the excess return on the market portfolio w_m.

Define

$$\mathbf{SR}_m = \frac{r_m - r_f}{\sigma}$$

as the **Sharpe-ratio** on the market portfolio w_m.

Let δ_m be the value of the risk aversion parameter that induces an investor to hold the market portfolio in light of the optimal portfolio choice rule (2).

Evidently, portfolio rule (2) then implies that $r_m - r_f = \delta_m \sigma^2$ or

$$\delta_m = \frac{r_m - r_f}{\sigma^2}$$

or

$$\delta_m = \frac{\mathbf{SR}_m}{\sigma}$$

Following the Black-Litterman philosophy, our first step will be to back a value of δ_m from

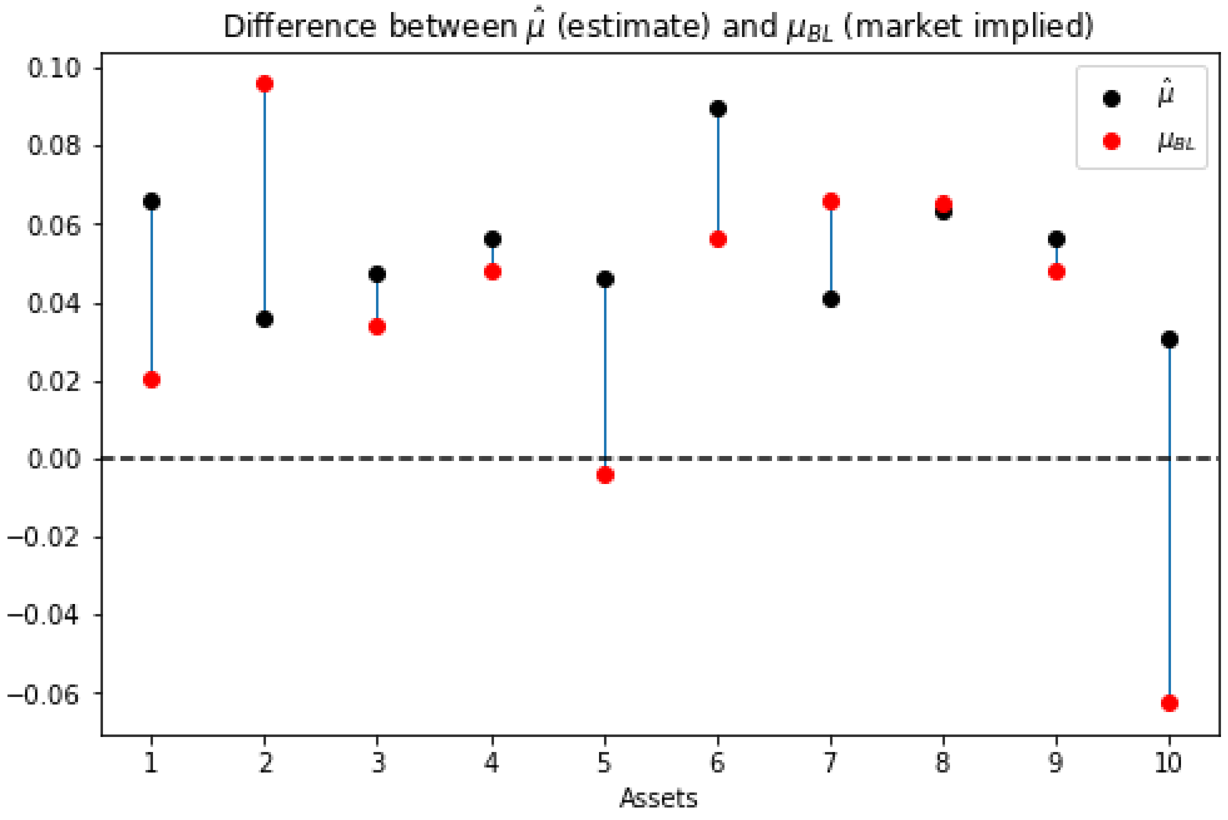
- an estimate of the Sharpe-ratio, and
- our maximum likelihood estimate of σ drawn from our estimates or w_m and Σ

The second key Black-Litterman step is then to use this value of δ together with the maximum likelihood estimate of Σ to deduce a $\mu_{\mathbf{BL}}$ that verifies portfolio rule (2) at the market portfolio $w = w_m$

$$\mu_m = \delta_m \Sigma w_m$$

The starting point of the Black-Litterman portfolio choice model is thus a pair (δ_m, μ_m) that tells the customer to hold the market portfolio.

```
# Observed mean excess market return
r_m = w_m @ μ_est

# Estimated variance of the market portfolio
σ_m = w_m @ Σ_est @ w_m

# Sharpe-ratio
sr_m = r_m / np.sqrt(σ_m)

# Risk aversion of market portfolio holder
d_m = r_m / σ_m

# Derive "view" which would induce the market portfolio
μ_m = (d_m * Σ_est @ w_m).reshape(N, 1)

fig, ax = plt.subplots(figsize=(8, 5))
ax.set_title(r'Difference between $\hat{\mu}$ (estimate) and $\mu_{BL}$ (market␣
↪implied)')
ax.plot(np.arange(N)+1, μ_est, 'o', c='k', label='$\hat{\mu}$')
ax.plot(np.arange(N)+1, μ_m, 'o', c='r', label='$\mu_{BL}$')
ax.vlines(np.arange(N) + 1, μ_m, μ_est, lw=1)
ax.axhline(0, c='k', ls='--')
ax.set_xlabel('Assets')
ax.xaxis.set_ticks(np.arange(1, N+1, 1))
plt.legend(numpoints=1)
plt.show()
```



35.1.7 Adding *Views*

Black and Litterman start with a baseline customer who asserts that he or she shares the **market's views**, which means that he or she believes that excess returns are governed by

$$\vec{r} - r_f \mathbf{1} \sim \mathcal{N}(\mu_{BL}, \Sigma) \tag{3}$$

Black and Litterman would advise that customer to hold the market portfolio of risky securities.

Black and Litterman then imagine a consumer who would like to express a view that differs from the market's.

The consumer wants appropriately to mix his view with the market's before using (2) to choose a portfolio.

Suppose that the customer's view is expressed by a hunch that rather than (3), excess returns are governed by

$$\vec{r} - r_f \mathbf{1} \sim \mathcal{N}(\hat{\mu}, \tau\Sigma)$$

where $\tau > 0$ is a scalar parameter that determines how the decision maker wants to mix his view $\hat{\mu}$ with the market's view $\mu_{\mathbf{BL}}$.

Black and Litterman would then use a formula like the following one to mix the views $\hat{\mu}$ and $\mu_{\mathbf{BL}}$

$$\tilde{\mu} = (\Sigma^{-1} + (\tau\Sigma)^{-1})^{-1}(\Sigma^{-1}\mu_{BL} + (\tau\Sigma)^{-1}\hat{\mu}) \tag{4}$$

Black and Litterman would then advise the customer to hold the portfolio associated with these views implied by rule (2):

$$\tilde{w} = (\delta\Sigma)^{-1}\tilde{\mu}$$

This portfolio $\tilde{w}$ will deviate from the portfolio w_{BL} in amounts that depend on the mixing parameter τ.

If $\hat{\mu}$ is the maximum likelihood estimator and τ is chosen heavily to weight this view, then the customer's portfolio will involve big short-long positions.

```
def black_litterman(λ, μ1, μ2, Σ1, Σ2):
    """
    This function calculates the Black-Litterman mixture
    mean excess return and covariance matrix
    """
    Σ1_inv = np.linalg.inv(Σ1)
    Σ2_inv = np.linalg.inv(Σ2)

    μ_tilde = np.linalg.solve(Σ1_inv + λ * Σ2_inv,
                              Σ1_inv @ μ1 + λ * Σ2_inv @ μ2)
    return μ_tilde

τ = 1
μ_tilde = black_litterman(1, μ_m, μ_est, Σ_est, τ * Σ_est)

# The Black-Litterman recommendation for the portfolio weights
w_tilde = np.linalg.solve(δ * Σ_est, μ_tilde)

τ_slider = FloatSlider(min=0.05, max=10, step=0.5, value=τ)

@interact(τ=τ_slider)
def BL_plot(τ):
    μ_tilde = black_litterman(1, μ_m, μ_est, Σ_est, τ * Σ_est)
    w_tilde = np.linalg.solve(δ * Σ_est, μ_tilde)

    fig, ax = plt.subplots(1, 2, figsize=(16, 6))
    ax[0].plot(np.arange(N)+1, μ_est, 'o', c='k',
               label=r'$\hat{\mu}$ (subj view)')
    ax[0].plot(np.arange(N)+1, μ_m, 'o', c='r',
               label=r'$\mu_{BL}$ (market)')
    ax[0].plot(np.arange(N)+1, μ_tilde, 'o', c='y',
               label=r'$\tilde{\mu}$ (mixture)')
    ax[0].vlines(np.arange(N)+1, μ_m, μ_est, lw=1)
    ax[0].axhline(0, c='k', ls='--')
    ax[0].set(xlim=(0, N+1), xlabel='Assets',
             title=r'Relationship between $\hat{\mu}$, $\mu_{BL}$, and  $ \tilde{\mu}
↪$')
    ax[0].xaxis.set_ticks(np.arange(1, N+1, 1))
    ax[0].legend(numpoints=1)

    ax[1].set_title('Black-Litterman portfolio weight recommendation')
    ax[1].plot(np.arange(N)+1, w, 'o', c='k', label=r'$w$ (mean-variance)')
    ax[1].plot(np.arange(N)+1, w_m, 'o', c='r', label=r'$w_{m}$ (market, BL)')
    ax[1].plot(np.arange(N)+1, w_tilde, 'o', c='y',
               label=r'$\tilde{w}$ (mixture)')
    ax[1].vlines(np.arange(N)+1, 0, w, lw=1)
    ax[1].vlines(np.arange(N)+1, 0, w_m, lw=1)
    ax[1].axhline(0, c='k')
    ax[1].axhline(-1, c='k', ls='--')
    ax[1].axhline(1, c='k', ls='--')
    ax[1].set(xlim=(0, N+1), xlabel='Assets',
             title='Black-Litterman portfolio weight recommendation')
    ax[1].xaxis.set_ticks(np.arange(1, N+1, 1))
    ax[1].legend(numpoints=1)
    plt.show()
```

```
interactive(children=(FloatSlider(value=1.0, description='τ', max=10.0, min=0.05,␣
↪step=0.5), Output()), _dom_c…
```

35.1.8 Bayes Interpretation of the Black-Litterman Recommendation

Consider the following Bayesian interpretation of the Black-Litterman recommendation.

The prior belief over the mean excess returns is consistent with the market portfolio and is given by

$$\mu \sim \mathcal{N}(\mu_{BL}, \Sigma)$$

Given a particular realization of the mean excess returns μ one observes the average excess returns $\hat{\mu}$ on the market according to the distribution

$$\hat{\mu} \mid \mu, \Sigma \sim \mathcal{N}(\mu, \tau\Sigma)$$

where τ is typically small capturing the idea that the variation in the mean is smaller than the variation of the individual random variable.

Given the realized excess returns one should then update the prior over the mean excess returns according to Bayes rule.

The corresponding posterior over mean excess returns is normally distributed with mean

$$(\Sigma^{-1} + (\tau\Sigma)^{-1})^{-1}(\Sigma^{-1}\mu_{BL} + (\tau\Sigma)^{-1}\hat{\mu})$$

The covariance matrix is

$$(\Sigma^{-1} + (\tau\Sigma)^{-1})^{-1}$$

Hence, the Black-Litterman recommendation is consistent with the Bayes update of the prior over the mean excess returns in light of the realized average excess returns on the market.

35.1.9 Curve Decolletage

Consider two independent "competing" views on the excess market returns

$$\vec{r}_e \sim \mathcal{N}(\mu_{BL}, \Sigma)$$

and

$$\vec{r}_e \sim \mathcal{N}(\hat{\mu}, \tau\Sigma)$$

A special feature of the multivariate normal random variable Z is that its density function depends only on the (Euclidiean) length of its realization z.

Formally, let the k-dimensional random vector be

$$Z \sim \mathcal{N}(\mu, \Sigma)$$

then

$$\bar{Z} \equiv \Sigma(Z - \mu) \sim \mathcal{N}(\mathbf{0}, I)$$

and so the points where the density takes the same value can be described by the ellipse

$$\bar{z} \cdot \bar{z} = (z - \mu)'\Sigma^{-1}(z - \mu) = \bar{d} \tag{5}$$

where $\bar{d} \in \mathbb{R}_+$ denotes the (transformation) of a particular density value.

The curves defined by equation (5) can be labeled as iso-likelihood ellipses

Remark: More generally there is a class of density functions that possesses this feature, i.e.

$$\exists g : \mathbb{R}_+ \mapsto \mathbb{R}_+ \quad \text{and} \quad c \geq 0, \quad \text{s.t. the density} \quad f \quad \text{of} \quad Z \quad \text{has the form} \qquad f(z) = cg(z \cdot z)$$

This property is called **spherical symmetry** (see p 81. in Leamer (1978) [Lea78]).

In our specific example, we can use the pair $(\bar{d}_1, \bar{d}_2)$ as being two "likelihood" values for which the corresponding iso-likelihood ellipses in the excess return space are given by

$$(\vec{r}_e - \mu_{BL})'\Sigma^{-1}(\vec{r}_e - \mu_{BL}) = \bar{d}_1$$
$$(\vec{r}_e - \hat{\mu})' \left(\tau\Sigma\right)^{-1} (\vec{r}_e - \hat{\mu}) = \bar{d}_2$$

Notice that for particular $\bar{d}_1$ and $\bar{d}_2$ values the two ellipses have a tangency point.

These tangency points, indexed by the pairs $(\bar{d}_1, \bar{d}_2)$, characterize points $\vec{r}_e$ from which there exists no deviation where one can increase the likelihood of one view without decreasing the likelihood of the other view.

The pairs $(\bar{d}_1, \bar{d}_2)$ for which there is such a point outlines a curve in the excess return space. This curve is reminiscent of the Pareto curve in an Edgeworth-box setting.

Dickey (1975) [Dic75] calls it a *curve decolletage*.

Leamer (1978) [Lea78] calls it an *information contract curve* and describes it by the following program: maximize the likelihood of one view, say the Black-Litterman recommendation while keeping the likelihood of the other view at least at a prespecified constant $\bar{d}_2$

$$\bar{d}_1(\bar{d}_2) \equiv \max_{\vec{r}_e} \ (\vec{r}_e - \mu_{BL})'\Sigma^{-1}(\vec{r}_e - \mu_{BL})$$
$$\text{subject to} \quad (\vec{r}_e - \hat{\mu})'(\tau\Sigma)^{-1}(\vec{r}_e - \hat{\mu}) \geq \bar{d}_2$$

Denoting the multiplier on the constraint by λ, the first-order condition is

$$2(\vec{r}_e - \mu_{BL})'\Sigma^{-1} + \lambda 2(\vec{r}_e - \hat{\mu})'(\tau\Sigma)^{-1} = \mathbf{0}$$

which defines the *information contract curve* between μ_{BL} and $\hat{\mu}$

$$\vec{r}_e = (\Sigma^{-1} + \lambda(\tau\Sigma)^{-1})^{-1}(\Sigma^{-1}\mu_{BL} + \lambda(\tau\Sigma)^{-1}\hat{\mu}) \tag{6}$$

Note that if $\lambda = 1$, (6) is equivalent with (4) and it identifies one point on the information contract curve.

Furthermore, because λ is a function of the minimum likelihood $\bar{d}_2$ on the RHS of the constraint, by varying $\bar{d}_2$ (or λ), we can trace out the whole curve as the figure below illustrates.

```python
np.random.seed(1987102)

N = 2                                               # Number of assets
T = 200                                             # Sample size
τ = 0.8

# Random market portfolio (sum is normalized to 1)
w_m = np.random.rand(N)
w_m = w_m / (w_m.sum())

μ = (np.random.randn(N) + 5) / 100
S = np.random.randn(N, N)
V = S @ S.T
Σ = V * (w_m @ μ)**2 / (w_m @ V @ w_m)

excess_return = stat.multivariate_normal(μ, Σ)
```

(continues on next page)

(continued from previous page)

```
sample = excess_return.rvs(T)

μ_est = sample.mean(0).reshape(N, 1)
Σ_est = np.cov(sample.T)

σ_m = w_m @ Σ_est @ w_m
d_m = (w_m @ μ_est) / σ_m
μ_m = (d_m * Σ_est @ w_m).reshape(N, 1)

N_r1, N_r2 = 100, 100
r1 = np.linspace(-0.04, .1, N_r1)
r2 = np.linspace(-0.02, .15, N_r2)

λ_grid = np.linspace(.001, 20, 100)
curve = np.asarray([black_litterman(λ, μ_m, μ_est, Σ_est,
                                    τ * Σ_est).flatten() for λ in λ_grid])

λ_slider = FloatSlider(min=.1, max=7, step=.5, value=1)

@interact(λ=λ_slider)
def decolletage(λ):
    dist_r_BL = stat.multivariate_normal(μ_m.squeeze(), Σ_est)
    dist_r_hat = stat.multivariate_normal(μ_est.squeeze(), τ * Σ_est)

    X, Y = np.meshgrid(r1, r2)
    Z_BL = np.zeros((N_r1, N_r2))
    Z_hat = np.zeros((N_r1, N_r2))

    for i in range(N_r1):
        for j in range(N_r2):
            Z_BL[i, j] = dist_r_BL.pdf(np.hstack([X[i, j], Y[i, j]]))
            Z_hat[i, j] = dist_r_hat.pdf(np.hstack([X[i, j], Y[i, j]]))

    μ_tilde = black_litterman(λ, μ_m, μ_est, Σ_est, τ * Σ_est).flatten()

    fig, ax = plt.subplots(figsize=(10, 6))
    ax.contourf(X, Y, Z_hat, cmap='viridis', alpha =.4)
    ax.contourf(X, Y, Z_BL, cmap='viridis', alpha =.4)
    ax.contour(X, Y, Z_BL, [dist_r_BL.pdf(μ_tilde)], cmap='viridis', alpha=.9)
    ax.contour(X, Y, Z_hat, [dist_r_hat.pdf(μ_tilde)], cmap='viridis', alpha=.9)
    ax.scatter(μ_est[0], μ_est[1])
    ax.scatter(μ_m[0], μ_m[1])
    ax.scatter(μ_tilde[0], μ_tilde[1], c='k', s=20*3)

    ax.plot(curve[:, 0], curve[:, 1], c='k')
    ax.axhline(0, c='k', alpha=.8)
    ax.axvline(0, c='k', alpha=.8)
    ax.set_xlabel(r'Excess return on the first asset, $r_{e, 1}$')
    ax.set_ylabel(r'Excess return on the second asset, $r_{e, 2}$')
    ax.text(μ_est[0] + 0.003, μ_est[1], r'$\hat{\mu}$')
    ax.text(μ_m[0] + 0.003, μ_m[1] + 0.005, r'$\mu_{BL}$')
    plt.show()
```

```
interactive(children=(FloatSlider(value=1.0, description='λ', max=7.0, min=0.1,␣
↪step=0.5), Output()), _dom_cla…
```

Note that the line that connects the two points $\hat{\mu}$ and μ_{BL} is linear, which comes from the fact that the covariance matrices of the two competing distributions (views) are proportional to each other.

To illustrate the fact that this is not necessarily the case, consider another example using the same parameter values, except that the "second view" constituting the constraint has covariance matrix τI instead of $\tau\Sigma$.

This leads to the following figure, on which the curve connecting $\hat{\mu}$ and μ_{BL} are bending

```python
λ_grid = np.linspace(.001, 20000, 1000)
curve = np.asarray([black_litterman(λ, μ_m, μ_est, Σ_est,
                                    τ * np.eye(N)).flatten() for λ in λ_grid])

λ_slider = FloatSlider(min=5, max=1500, step=100, value=200)

@interact(λ=λ_slider)
def decolletage(λ):
    dist_r_BL = stat.multivariate_normal(μ_m.squeeze(), Σ_est)
    dist_r_hat = stat.multivariate_normal(μ_est.squeeze(), τ * np.eye(N))

    X, Y = np.meshgrid(r1, r2)
    Z_BL = np.zeros((N_r1, N_r2))
    Z_hat = np.zeros((N_r1, N_r2))

    for i in range(N_r1):
        for j in range(N_r2):
            Z_BL[i, j] = dist_r_BL.pdf(np.hstack([X[i, j], Y[i, j]]))
            Z_hat[i, j] = dist_r_hat.pdf(np.hstack([X[i, j], Y[i, j]]))

    μ_tilde = black_litterman(λ, μ_m, μ_est, Σ_est, τ * np.eye(N)).flatten()

    fig, ax = plt.subplots(figsize=(10, 6))
    ax.contourf(X, Y, Z_hat, cmap='viridis', alpha=.4)
    ax.contourf(X, Y, Z_BL, cmap='viridis', alpha=.4)
    ax.contour(X, Y, Z_BL, [dist_r_BL.pdf(μ_tilde)], cmap='viridis', alpha=.9)
    ax.contour(X, Y, Z_hat, [dist_r_hat.pdf(μ_tilde)], cmap='viridis', alpha=.9)
    ax.scatter(μ_est[0], μ_est[1])
    ax.scatter(μ_m[0], μ_m[1])

    ax.scatter(μ_tilde[0], μ_tilde[1], c='k', s=20*3)

    ax.plot(curve[:, 0], curve[:, 1], c='k')
    ax.axhline(0, c='k', alpha=.8)
    ax.axvline(0, c='k', alpha=.8)
    ax.set_xlabel(r'Excess return on the first asset, $r_{e, 1}$')
    ax.set_ylabel(r'Excess return on the second asset, $r_{e, 2}$')
    ax.text(μ_est[0] + 0.003, μ_est[1], r'$\hat{\mu}$')
    ax.text(μ_m[0] + 0.003, μ_m[1] + 0.005, r'$\mu_{BL}$')
    plt.show()
```

```
interactive(children=(FloatSlider(value=200.0, description='λ', max=1500.0, min=5.0,␣
↪step=100.0), Output()), _…
```

35.1.10 Black-Litterman Recommendation as Regularization

First, consider the OLS regression

$$\min_{\beta} \|X\beta - y\|^2$$

which yields the solution

$$\hat{\beta}_{OLS} = (X'X)^{-1}X'y$$

A common performance measure of estimators is the *mean squared error (MSE)*.

An estimator is "good" if its MSE is relatively small. Suppose that β_0 is the "true" value of the coefficient, then the MSE of the OLS estimator is

$$\text{mse}(\hat{\beta}_{OLS}, \beta_0) := \mathbb{E}\|\hat{\beta}_{OLS} - \beta_0\|^2 = \underbrace{\mathbb{E}\|\hat{\beta}_{OLS} - \mathbb{E}\hat{\beta}_{OLS}\|^2}_{\text{variance}} + \underbrace{\|\mathbb{E}\hat{\beta}_{OLS} - \beta_0\|^2}_{\text{bias}}$$

From this decomposition, one can see that in order for the MSE to be small, both the bias and the variance terms must be small.

For example, consider the case when X is a T-vector of ones (where T is the sample size), so $\hat{\beta}_{OLS}$ is simply the sample average, while $\beta_0 \in \mathbb{R}$ is defined by the true mean of y.

In this example the MSE is

$$\text{mse}(\hat{\beta}_{OLS}, \beta_0) = \underbrace{\frac{1}{T^2}\mathbb{E}\left(\sum_{t=1}^{T}(y_t - \beta_0)\right)^2}_{\text{variance}} + \underbrace{0}_{\text{bias}}$$

However, because there is a trade-off between the estimator's bias and variance, there are cases when by permitting a small bias we can substantially reduce the variance so overall the MSE gets smaller.

A typical scenario when this proves to be useful is when the number of coefficients to be estimated is large relative to the sample size.

In these cases, one approach to handle the bias-variance trade-off is the so called *Tikhonov regularization*.

A general form with regularization matrix Γ can be written as

$$\min_{\beta} \left\{ \|X\beta - y\|^2 + \|\Gamma(\beta - \tilde{\beta})\|^2 \right\}$$

which yields the solution

$$\hat{\beta}_{Reg} = (X'X + \Gamma'\Gamma)^{-1}(X'y + \Gamma'\Gamma\tilde{\beta})$$

Substituting the value of $\hat{\beta}_{OLS}$ yields

$$\hat{\beta}_{Reg} = (X'X + \Gamma'\Gamma)^{-1}(X'X\hat{\beta}_{OLS} + \Gamma'\Gamma\tilde{\beta})$$

Often, the regularization matrix takes the form $\Gamma = \lambda I$ with $\lambda > 0$ and $\tilde{\beta} = \mathbf{0}$.

Then the Tikhonov regularization is equivalent to what is called *ridge regression* in statistics.

To illustrate how this estimator addresses the bias-variance trade-off, we compute the MSE of the ridge estimator

$$\text{mse}(\hat{\beta}_{\text{ridge}}, \beta_0) = \underbrace{\frac{1}{(T+\lambda)^2}\mathbb{E}\left(\sum_{t=1}^{T}(y_t - \beta_0)\right)^2}_{\text{variance}} + \underbrace{\left(\frac{\lambda}{T+\lambda}\right)^2 \beta_0^2}_{\text{bias}}$$

The ridge regression shrinks the coefficients of the estimated vector towards zero relative to the OLS estimates thus reducing the variance term at the cost of introducing a "small" bias.

However, there is nothing special about the zero vector.

When $\tilde{\beta} \neq \mathbf{0}$ shrinkage occurs in the direction of $\tilde{\beta}$.

Now, we can give a regularization interpretation of the Black-Litterman portfolio recommendation.

To this end, simplify first the equation (4) characterizing the Black-Litterman recommendation

$$\begin{aligned}\tilde{\mu} &= (\Sigma^{-1} + (\tau\Sigma)^{-1})^{-1}(\Sigma^{-1}\mu_{BL} + (\tau\Sigma)^{-1}\hat{\mu}) \\ &= (1+\tau^{-1})^{-1}\Sigma\Sigma^{-1}(\mu_{BL} + \tau^{-1}\hat{\mu}) \\ &= (1+\tau^{-1})^{-1}(\mu_{BL} + \tau^{-1}\hat{\mu})\end{aligned}$$

In our case, $\hat{\mu}$ is the estimated mean excess returns of securities. This could be written as a vector autoregression where

- y is the stacked vector of observed excess returns of size $(NT \times 1)$ – N securities and T observations.
- $X = \sqrt{T^{-1}}(I_N \otimes \iota_T)$ where I_N is the identity matrix and ι_T is a column vector of ones.

Correspondingly, the OLS regression of y on X would yield the mean excess returns as coefficients.

With $\Gamma = \sqrt{\tau T^{-1}}(I_N \otimes \iota_T)$ we can write the regularized version of the mean excess return estimation

$$\begin{aligned}\hat{\beta}_{Reg} &= (X'X + \Gamma'\Gamma)^{-1}(X'X\hat{\beta}_{OLS} + \Gamma'\Gamma\tilde{\beta}) \\ &= (1+\tau)^{-1}X'X(X'X)^{-1}(\hat{\beta}_{OLS} + \tau\tilde{\beta}) \\ &= (1+\tau)^{-1}(\hat{\beta}_{OLS} + \tau\tilde{\beta}) \\ &= (1+\tau^{-1})^{-1}(\tau^{-1}\hat{\beta}_{OLS} + \tilde{\beta})\end{aligned}$$

Given that $\hat{\beta}_{OLS} = \hat{\mu}$ and $\tilde{\beta} = \mu_{BL}$ in the Black-Litterman model, we have the following interpretation of the model's recommendation.

The estimated (personal) view of the mean excess returns, $\hat{\mu}$ that would lead to extreme short-long positions are "shrunk" towards the conservative market view, μ_{BL}, that leads to the more conservative market portfolio.

So the Black-Litterman procedure results in a recommendation that is a compromise between the conservative market portfolio and the more extreme portfolio that is implied by estimated "personal" views.

35.1.11 Digression on A Robust Control Operator

The Black-Litterman approach is partly inspired by the econometric insight that it is easier to estimate covariances of excess returns than the means.

That is what gave Black and Litterman license to adjust investors' perception of mean excess returns while not tampering with the covariance matrix of excess returns.

The robust control theory is another approach that also hinges on adjusting mean excess returns but not covariances.

Associated with a robust control problem is what Hansen and Sargent [HS01], [HS08a] call a T operator.

Let's define the T operator as it applies to the problem at hand.

Let x be an $n \times 1$ Gaussian random vector with mean vector μ and covariance matrix $\Sigma = CC'$. This means that x can be represented as

$$x = \mu + C\epsilon$$

where $\epsilon \sim \mathcal{N}(0, I)$.

Let $\phi(\epsilon)$ denote the associated standardized Gaussian density.

Let $m(\epsilon, \mu)$ be a **likelihood ratio**, meaning that it satisfies

- $m(\epsilon, \mu) > 0$
- $\int m(\epsilon, \mu)\phi(\epsilon)d\epsilon = 1$

That is, $m(\epsilon, \mu)$ is a non-negative random variable with mean 1.

Multiplying $\phi(\epsilon)$ by the likelihood ratio $m(\epsilon, \mu)$ produces a distorted distribution for ϵ, namely

$$\tilde{\phi}(\epsilon) = m(\epsilon, \mu)\phi(\epsilon)$$

The next concept that we need is the **entropy** of the distorted distribution $\tilde{\phi}$ with respect to ϕ.

Entropy is defined as

$$\text{ent} = \int \log m(\epsilon, \mu)m(\epsilon, \mu)\phi(\epsilon)d\epsilon$$

or

$$\text{ent} = \int \log m(\epsilon, \mu)\tilde{\phi}(\epsilon)d\epsilon$$

That is, relative entropy is the expected value of the likelihood ratio m where the expectation is taken with respect to the twisted density $\tilde{\phi}$.

Relative entropy is non-negative. It is a measure of the discrepancy between two probability distributions.

As such, it plays an important role in governing the behavior of statistical tests designed to discriminate one probability distribution from another.

We are ready to define the T operator.

Let $V(x)$ be a value function.

Define

$$\begin{aligned}\mathsf{T}\left(V(x)\right) &= \min_{m(\epsilon,\mu)} \int m(\epsilon, \mu)[V(\mu + C\epsilon) + \theta \log m(\epsilon, \mu)]\phi(\epsilon)d\epsilon \\ &= -\log\theta \int \exp\left(\frac{-V(\mu + C\epsilon)}{\theta}\right)\phi(\epsilon)d\epsilon\end{aligned}$$

This asserts that T is an indirect utility function for a minimization problem in which an **evil agent** chooses a distorted probability distribution $\tilde{\phi}$ to lower expected utility, subject to a penalty term that gets bigger the larger is relative entropy.

Here the penalty parameter

$$\theta \in [\underline{\theta}, +\infty]$$

is a robustness parameter when it is $+\infty$, there is no scope for the minimizing agent to distort the distribution, so no robustness to alternative distributions is acquired As θ is lowered, more robustness is achieved.

Note: The T operator is sometimes called a *risk-sensitivity* operator.

We shall apply $\mathsf{T}$to the special case of a linear value function $w'(\vec{r} - r_f 1)$ where $\vec{r} - r_f 1 \sim \mathcal{N}(\mu, \Sigma)$ or $\vec{r} - r_f \mathbf{1} = \mu + C\epsilon$and $\epsilon \sim \mathcal{N}(0, I)$.

The associated worst-case distribution of ϵ is Gaussian with mean $v = -\theta^{-1}C'w$ and covariance matrix I (When the value function is affine, the worst-case distribution distorts the mean vector of ϵ but not the covariance matrix of ϵ).

For utility function argument $w'(\vec{r} - r_f 1)$

$$\mathsf{T}(\vec{r} - r_f \mathbf{1}) = w'\mu + \zeta - \frac{1}{2\theta} w'\Sigma w$$

and entropy is

$$\frac{v'v}{2} = \frac{1}{2\theta^2} w' C C' w$$

35.1.12 A Robust Mean-variance Portfolio Model

According to criterion (1), the mean-variance portfolio choice problem chooses w to maximize

$$E[w(\vec{r} - r_f \mathbf{1})]] - \text{var}[w(\vec{r} - r_f \mathbf{1})]$$

which equals

$$w'\mu - \frac{\delta}{2} w'\Sigma w$$

A robust decision maker can be modeled as replacing the mean return $E[w(\vec{r} - r_f \mathbf{1})]$ with the risk-sensitive

$$\mathsf{T}[w(\vec{r} - r_f \mathbf{1})] = w'\mu - \frac{1}{2\theta} w'\Sigma w$$

that comes from replacing the mean μ of $\vec{r} - r\_f \mathbf{1}$ with the worst-case mean

$$\mu - \theta^{-1} \Sigma w$$

Notice how the worst-case mean vector depends on the portfolio w.

The operator T is the indirect utility function that emerges from solving a problem in which an agent who chooses probabilities does so in order to minimize the expected utility of a maximizing agent (in our case, the maximizing agent chooses portfolio weights w).

The robust version of the mean-variance portfolio choice problem is then to choose a portfolio w that maximizes

$$\mathsf{T}[w(\vec{r} - r_f \mathbf{1})] - \frac{\delta}{2} w'\Sigma w$$

or

$$w'(\mu - \theta^{-1}\Sigma w) - \frac{\delta}{2} w'\Sigma w \tag{7}$$

The minimizer of (7) is

$$w_{\text{rob}} = \frac{1}{\delta + \gamma} \Sigma^{-1} \mu$$

where $\gamma \equiv \theta^{-1}$ is sometimes called the risk-sensitivity parameter.

An increase in the risk-sensitivity parameter γ shrinks the portfolio weights toward zero in the same way that an increase in risk aversion does.

35.2 Appendix

We want to illustrate the "folk theorem" that with high or moderate frequency data, it is more difficult to estimate means than variances.

In order to operationalize this statement, we take two analog estimators:

- sample average: $\bar{X}_N = \frac{1}{N}\sum_{i=1}^N X_i$
- sample variance: $S_N = \frac{1}{N-1}\sum_{t=1}^N (X_i - \bar{X}_N)^2$

to estimate the unconditional mean and unconditional variance of the random variable X, respectively.

To measure the "difficulty of estimation", we use *mean squared error* (MSE), that is the average squared difference between the estimator and the true value.

Assuming that the process $\{X_i\}$is ergodic, both analog estimators are known to converge to their true values as the sample size N goes to infinity.

More precisely for all $\varepsilon > 0$

$$\lim_{N\to\infty} \quad P\left\{\left|\bar{X}_N - \mathbb{E}X\right| > \varepsilon\right\} = 0$$

and

$$\lim_{N\to\infty} \quad P\left\{\left|S_N - \mathbb{V}X\right| > \varepsilon\right\} = 0$$

A necessary condition for these convergence results is that the associated MSEs vanish as N goes to infinity, or in other words,

$$\text{MSE}(\bar{X}_N, \mathbb{E}X) = o(1) \qquad \text{and} \qquad \text{MSE}(S_N, \mathbb{V}X) = o(1)$$

Even if the MSEs converge to zero, the associated rates might be different. Looking at the limit of the *relative MSE* (as the sample size grows to infinity)

$$\frac{\text{MSE}(S_N, \mathbb{V}X)}{\text{MSE}(\bar{X}_N, \mathbb{E}X)} = \frac{o(1)}{o(1)} \underset{N\to\infty}{\to} B$$

can inform us about the relative (asymptotic) rates.

We will show that in general, with dependent data, the limit B depends on the sampling frequency.

In particular, we find that the rate of convergence of the variance estimator is less sensitive to increased sampling frequency than the rate of convergence of the mean estimator.

Hence, we can expect the relative asymptotic rate, B, to get smaller with higher frequency data, illustrating that "it is more difficult to estimate means than variances".

That is, we need significantly more data to obtain a given precision of the mean estimate than for our variance estimate.

35.2.1 A Special Case – IID Sample

We start our analysis with the benchmark case of IID data. Consider a sample of size N generated by the following IID process,

$$X_i \sim \mathcal{N}(\mu, \sigma^2)$$

Taking $\bar{X}_N$ to estimate the mean, the MSE is

$$\text{MSE}(\bar{X}_N, \mu) = \frac{\sigma^2}{N}$$

Taking S_N to estimate the variance, the MSE is

$$\text{MSE}(S_N, \sigma^2) = \frac{2\sigma^4}{N-1}$$

Both estimators are unbiased and hence the MSEs reflect the corresponding variances of the estimators.

Furthermore, both MSEs are $o(1)$ with a (multiplicative) factor of difference in their rates of convergence:

$$\frac{\text{MSE}(S_N, \sigma^2)}{\text{MSE}(\bar{X}_N, \mu)} = \frac{N2\sigma^2}{N-1} \quad \underset{N\to\infty}{\to} \quad 2\sigma^2$$

We are interested in how this (asymptotic) relative rate of convergence changes as increasing sampling frequency puts dependence into the data.

35.2.2 Dependence and Sampling Frequency

To investigate how sampling frequency affects relative rates of convergence, we assume that the data are generated by a mean-reverting continuous time process of the form

$$dX_t = -\kappa(X_t - \mu)dt + \sigma dW_t$$

where $\mu$is the unconditional mean, $\kappa > 0$ is a persistence parameter, and $\{W_t\}$ is a standardized Brownian motion.

Observations arising from this system in particular discrete periods $\mathcal{T}(h) \equiv \{nh : n \in \mathbb{Z}\}$with$h > 0$ can be described by the following process

$$X_{t+1} = (1 - \exp(-\kappa h))\mu + \exp(-\kappa h)X_t + \epsilon_{t,h}$$

where

$$\epsilon_{t,h} \sim \mathcal{N}(0, \Sigma_h) \quad \text{with} \quad \Sigma_h = \frac{\sigma^2(1 - \exp(-2\kappa h))}{2\kappa}$$

We call h the *frequency* parameter, whereas n represents the number of *lags* between observations.

Hence, the effective distance between two observations X_t and X_{t+n} in the discrete time notation is equal to $h \cdot n$ in terms of the underlying continuous time process.

Straightforward calculations show that the autocorrelation function for the stochastic process $\{X_t\}_{t\in\mathcal{T}(h)}$ is

$$\Gamma_h(n) \equiv \text{corr}(X_{t+hn}, X_t) = \exp(-\kappa hn)$$

and the auto-covariance function is

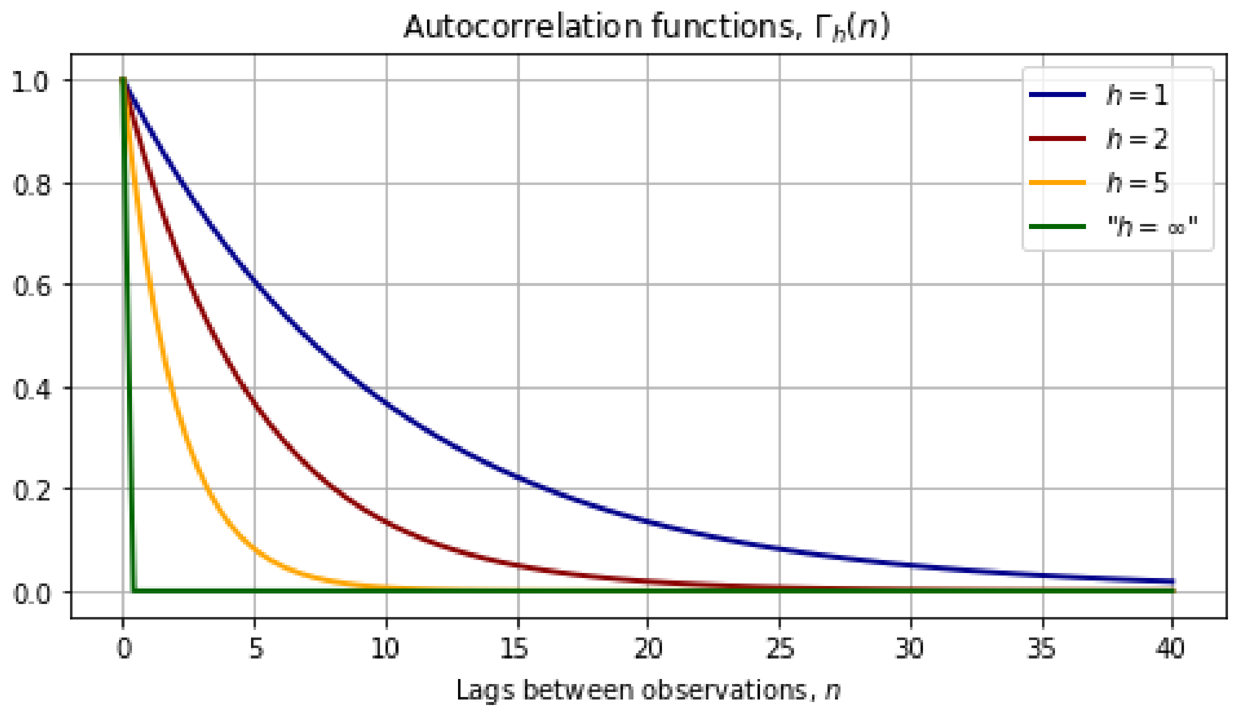
$$\gamma_h(n) \equiv \text{cov}(X_{t+hn}, X_t) = \frac{\exp(-\kappa hn)\sigma^2}{2\kappa}.$$

It follows that if $n = 0$, the unconditional variance is given by $\gamma_h(0) = \frac{\sigma^2}{2\kappa}$ irrespective of the sampling frequency.

The following figure illustrates how the dependence between the observations is related to the sampling frequency

- For any given h, the autocorrelation converges to zero as we increase the distance – n– between the observations. This represents the “weak dependence” of the X process.
- Moreover, for a fixed lag length, n, the dependence vanishes as the sampling frequency goes to infinity. In fact, letting h go to ∞ gives back the case of IID data.

```
μ = .0
κ = .1
σ = .5
var_uncond = σ**2 / (2 * κ)

n_grid = np.linspace(0, 40, 100)
autocorr_h1 = np.exp(-κ * n_grid * 1)
autocorr_h2 = np.exp(-κ * n_grid * 2)
autocorr_h5 = np.exp(-κ * n_grid * 5)
autocorr_h1000 = np.exp(-κ * n_grid * 1e8)

fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(n_grid, autocorr_h1, label=r'$h=1$', c='darkblue', lw=2)
ax.plot(n_grid, autocorr_h2, label=r'$h=2$', c='darkred', lw=2)
ax.plot(n_grid, autocorr_h5, label=r'$h=5$', c='orange', lw=2)
ax.plot(n_grid, autocorr_h1000, label=r'"$h=\infty$"', c='darkgreen', lw=2)
ax.legend()
ax.grid()
ax.set(title=r'Autocorrelation functions, $\Gamma_h(n)$',
       xlabel=r'Lags between observations, $n$')
plt.show()
```



35.2.3 Frequency and the Mean Estimator

Consider again the AR(1) process generated by discrete sampling with frequency h. Assume that we have a sample of size N and we would like to estimate the unconditional mean – in our case the true mean is μ.

Again, the sample average is an unbiased estimator of the unconditional mean

$$\mathbb{E}[\bar{X}_N] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[X_i] = \mathbb{E}[X_0] = \mu$$

The variance of the sample mean is given by

$$\begin{aligned}
\mathbb{V}\left(\bar{X}_N\right) &= \mathbb{V}\left(\frac{1}{N}\sum_{i=1}^{N}X_i\right) \\
&= \frac{1}{N^2}\left(\sum_{i=1}^{N}\mathbb{V}(X_i) + 2\sum_{i=1}^{N-1}\sum_{s=i+1}^{N}\text{cov}(X_i, X_s)\right) \\
&= \frac{1}{N^2}\left(N\gamma(0) + 2\sum_{i=1}^{N-1} i\cdot\gamma\left(h\cdot(N-i)\right)\right) \\
&= \frac{1}{N^2}\left(N\frac{\sigma^2}{2\kappa} + 2\sum_{i=1}^{N-1} i\cdot\exp(-\kappa h(N-i))\frac{\sigma^2}{2\kappa}\right)
\end{aligned}$$

It is explicit in the above equation that time dependence in the data inflates the variance of the mean estimator through the covariance terms. Moreover, as we can see, a higher sampling frequency—smaller h—makes all the covariance terms larger, everything else being fixed. This implies a relatively slower rate of convergence of the sample average for high-frequency data.

Intuitively, the stronger dependence across observations for high-frequency data reduces the "information content" of each observation relative to the IID case.

We can upper bound the variance term in the following way

$$\begin{aligned}
\mathbb{V}(\bar{X}_N) &= \frac{1}{N^2}\left(N\sigma^2 + 2\sum_{i=1}^{N-1} i\cdot\exp(-\kappa h(N-i))\sigma^2\right) \\
&\leq \frac{\sigma^2}{2\kappa N}\left(1 + 2\sum_{i=1}^{N-1}\cdot\exp(-\kappa h(i))\right) \\
&= \underbrace{\frac{\sigma^2}{2\kappa N}}_{\text{IID case}}\left(1 + 2\frac{1-\exp(-\kappa h)^{N-1}}{1-\exp(-\kappa h)}\right)
\end{aligned}$$

Asymptotically the $\exp(-\kappa h)^{N-1}$ vanishes and the dependence in the data inflates the benchmark IID variance by a factor of

$$\left(1 + 2\frac{1}{1-\exp(-\kappa h)}\right)$$

This long run factor is larger the higher is the frequency (the smaller is h).

Therefore, we expect the asymptotic relative MSEs, B, to change with time-dependent data. We just saw that the mean estimator's rate is roughly changing by a factor of

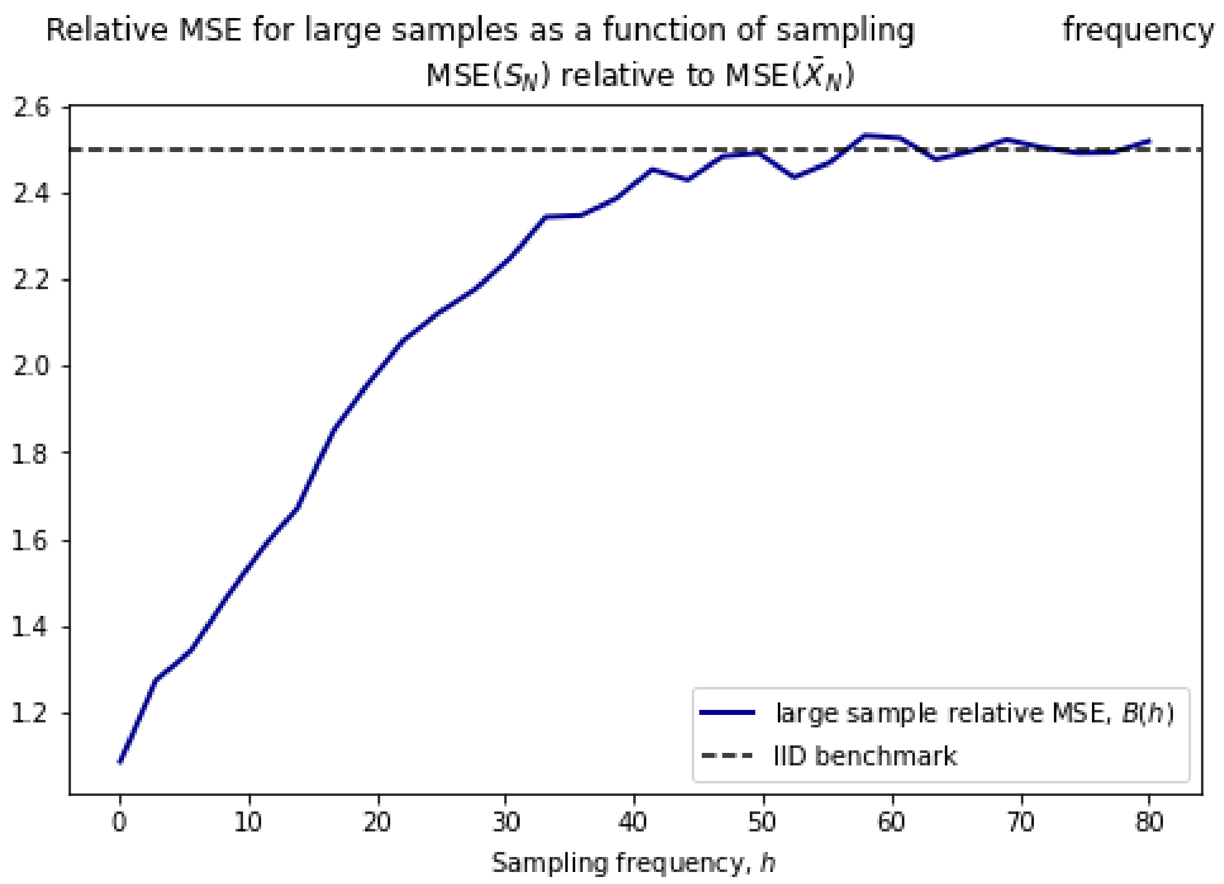
$$\left(1 + 2\frac{1}{1-\exp(-\kappa h)}\right)$$

Unfortunately, the variance estimator's MSE is harder to derive.

Nonetheless, we can approximate it by using (large sample) simulations, thus getting an idea about how the asymptotic relative MSEs changes in the sampling frequency h relative to the IID case that we compute in closed form.

```python
def sample_generator(h, N, M):
    ϕ = (1 - np.exp(-κ * h)) * μ
    ρ = np.exp(-κ * h)
    s = σ**2 * (1 - np.exp(-2 * κ * h)) / (2 * κ)

    mean_uncond = μ
    std_uncond = np.sqrt(σ**2 / (2 * κ))

    ε_path = stat.norm(0, np.sqrt(s)).rvs((M, N))

    y_path = np.zeros((M, N + 1))
    y_path[:, 0] = stat.norm(mean_uncond, std_uncond).rvs(M)

    for i in range(N):
        y_path[:, i + 1] = ϕ + ρ * y_path[:, i] + ε_path[:, i]

    return y_path
```

```python
# Generate large sample for different frequencies
N_app, M_app = 1000, 30000        # Sample size, number of simulations
h_grid = np.linspace(.1, 80, 30)

var_est_store = []
mean_est_store = []
labels = []

for h in h_grid:
    labels.append(h)
    sample = sample_generator(h, N_app, M_app)
    mean_est_store.append(np.mean(sample, 1))
    var_est_store.append(np.var(sample, 1))

var_est_store = np.array(var_est_store)
mean_est_store = np.array(mean_est_store)

# Save mse of estimators
mse_mean = np.var(mean_est_store, 1) + (np.mean(mean_est_store, 1) - μ)**2
mse_var = np.var(var_est_store, 1) \
          + (np.mean(var_est_store, 1) - var_uncond)**2

benchmark_rate = 2 * var_uncond       # IID case

# Relative MSE for large samples
rate_h = mse_var / mse_mean

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(h_grid, rate_h, c='darkblue', lw=2,
        label=r'large sample relative MSE, $B(h)$')
ax.axhline(benchmark_rate, c='k', ls='--', label=r'IID benchmark')
ax.set_title('Relative MSE for large samples as a function of sampling \
              frequency \n MSE($S_N$) relative to MSE($\\bar X_N$)')
ax.set_xlabel('Sampling frequency, $h$')
ax.legend()
plt.show()
```



The above figure illustrates the relationship between the asymptotic relative MSEs and the sampling frequency

- We can see that with low-frequency data – large values of h – the ratio of asymptotic rates approaches the IID case.
- As h gets smaller – the higher the frequency – the relative performance of the variance estimator is better in the sense that the ratio of asymptotic rates gets smaller. That is, as the time dependence gets more pronounced, the rate of convergence of the mean estimator's MSE deteriorates more than that of the variance estimator.

CHAPTER

THIRTYSIX

IRRELEVANCE OF CAPITAL STRUCTURE WITH COMPLETE MARKETS

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
!pip install interpolation
!conda install -y -c plotly plotly plotly-orca
```

36.1 Introduction

This is a prolegomenon to another lecture *Equilibrium Capital Structures with Incomplete Markets* about a model with incomplete markets authored by Bisin, Clementi, and Gottardi [BCG18].

We adopt specifications of preferences and technologies very close to Bisin, Clemente, and Gottardi's but unlike them assume that there are complete markets in one-period Arrow securities.

This simplification of BCG's setup helps us by

- creating a benchmark economy to compare with outcomes in BCG's incomplete markets economy
- creating a good guess for initial values of some equilibrium objects to be computed in BCG's incomplete markets economy via an iterative algorithm
- illustrating classic complete markets outcomes that include
  - indeterminacy of consumers' portfolio choices
  - indeterminacy of firms' financial structures that underlies a Modigliani-Miller theorem [MM58]
- introducing `Big K, little k` issues in a simple context that will recur in the BCG incomplete markets environment

A Big K, little k analysis also played roles in this quantecon lecture as well as here and *here*.

36.1.1 Setup

The economy lasts for two periods, $t = 0, 1$.

There are two types of consumers named $i = 1, 2$.

A scalar random variable ϵ with probability density $g(\epsilon)$ affects both

- the return in period 1 from investing $k \geq 0$ in physical capital in period 0.
- exogenous period 1 endowments of the consumption good for agents of types $i = 1$ and $i = 2$.

Type $i = 1$ and $i = 2$ agents' period 1 endowments are correlated with the return on physical capital in different ways.

We discuss two arrangements:

- a command economy in which a benevolent planner chooses k and allocates goods to the two types of consumers in each period and each random second period state
- a competitive equilibrium with markets in claims on physical capital and a complete set (possibly a continuum) of one-period Arrow securities that pay period 1 consumption goods contingent on the realization of random variable ϵ.

36.1.2 Endowments

There is a single consumption good in period 0 and at each random state ϵ in period 1.

Economy-wide endowments in periods 0 and 1 are

$$\begin{aligned} & w_0 \\ & w_1(\epsilon) \text{ in state } \epsilon \end{aligned}$$

Soon we'll explain how aggregate endowments are divided between type $i = 1$ and type $i = 2$ consumers.

We don't need to do that in order to describe a social planning problem.

36.1.3 Technology:

Where $\alpha \in (0, 1)$ and $A > 0$

$$\begin{aligned} c_0^1 + c_0^2 + k &= w_0^1 + w_0^2 \\ c_1^1(\epsilon) + c_1^2(\epsilon) &= w_1^1(\epsilon) + w_1^2(\epsilon) + e^{\epsilon} A k^{\alpha}, \quad k \geq 0 \end{aligned}$$

36.1.4 Preferences:

A consumer of type i orders period 0 consumption c_0^i and state ϵ, period 1 consumption $c_1^i(\epsilon)$ by

$$u^i = u(c_0^i) + \beta \int u(c_1^i(\epsilon)) g(\epsilon) d\epsilon, \quad i = 1, 2$$

$\beta \in (0, 1)$ and the one-period utility function is

$$u(c) = \begin{cases} \frac{c^{1-\gamma}}{1-\gamma} & \text{if } \gamma \neq 1 \\ \log c & \text{if } \gamma = 1 \end{cases}$$

36.1.5 Parameterizations

Following BCG, we shall employ the following parameterizations:

$$
\begin{aligned}
\epsilon &\sim \mathcal{N}(\mu, \sigma^2) \\
u(c) &= \frac{c^{1-\gamma}}{1-\gamma} \\
w_1^i(\epsilon) &= e^{-\chi_i \mu - .5\chi_i^2\sigma^2 + \chi_i \epsilon}, \quad \chi_i \in [0,1]
\end{aligned}
$$

Sometimes instead of asuming $\epsilon \sim g(\epsilon) = \mathcal{N}(0, \sigma^2)$, we'll assume that $g(\cdot)$ is a probability mass function that serves as a discrete approximation to a standardized normal density.

36.1.6 Pareto criterion and planning problem

The planner's objective function is

$$
\text{obj} = \phi_1 u^1 + \phi_2 u^2, \quad \phi_i \geq 0, \quad \phi_1 + \phi_2 = 1
$$

where $\phi_i \geq 0$ is a Pareto weight that the planner attaches to a consumer of type i.

We form the following Lagrangian for the planner's problem:

$$
\begin{aligned}
L = & \sum_{i=1}^{2} \phi_i \left[u(c_0^i) + \beta \int u(c_1^i(\epsilon)) g(\epsilon) d\epsilon \right] \\
& + \lambda_0 \left[w_0^1 + w_0^2 - k - c_0^1 - c_0^2 \right] \\
& + \beta \int \lambda_1(\epsilon) \left[w_1^1(\epsilon) + w_1^2(\epsilon) + e^\epsilon A k^\alpha - c_1^1(\epsilon) - c_1^2(\epsilon) \right] g(\epsilon) d\epsilon
\end{aligned}
$$

First-order necessary optimality conditions for the planning problem are:

$$
\begin{aligned}
c_0^1 : & \quad \phi_1 u'(c_0^1) - \lambda_0 = 0 \\
c_0^2 : & \quad \phi_2 u'(c_0^2) - \lambda_0 = 0 \\
c_1^1(\epsilon) : & \quad \phi_1 \beta u'(c_1^1(\epsilon)) g(\epsilon) - \beta \lambda_1(\epsilon) g(\epsilon) = 0 \\
c_1^2(\epsilon) : & \quad \phi_2 \beta u'(c_1^2(\epsilon)) g(\epsilon) - \beta \lambda_1(\epsilon) g(\epsilon) = 0 \\
k : & \quad -\lambda_0 + \beta \alpha A k^{\alpha-1} \int \lambda_1(\epsilon) e^\epsilon g(\epsilon) d\epsilon = 0
\end{aligned}
$$

The first four equations imply that

$$
\begin{aligned}
\frac{u'(c_1^1(\epsilon))}{u'(c_0^1))} = \frac{u'(c_1^2(\epsilon))}{u'(c_0^2))} &= \frac{\lambda_1(\epsilon)}{\lambda_0} \\
\frac{u'(c_0^1)}{u'(c_0^2)} = \frac{u'(c_1^1(\epsilon))}{u'(c_1^2(\epsilon))} &= \frac{\phi_2}{\phi_1}
\end{aligned}
$$

These together with the fifth first-order condition for the planner imply the following equation that determines an optimal choice of capital

$$
1 = \beta \alpha A k^{\alpha-1} \int \frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} e^\epsilon g(\epsilon) d\epsilon
$$

for $i = 1, 2$.

36.1.7 Helpful observations and bookkeeping

Evidently,

$$u'(c) = c^{-\gamma}$$

and

$$\frac{u'(c^1)}{u'(c^2)} = \left(\frac{c^1}{c^2}\right)^{-\gamma} = \frac{\phi_2}{\phi_1}$$

where it is to be understood that this equation holds for $c^1 = c_0^1$ and $c^2 = c_0^2$ and also for $c^1 = c^1(\epsilon)$ and $c^2 = c^2(\epsilon)$ for all ϵ.

With the same understanding, it follows that

$$\left(\frac{c^1}{c^2}\right) = \left(\frac{\phi_2}{\phi_1}\right)^{-\gamma^{-1}}$$

Let $c = c^1 + c^2$.

It follows from the preceding equation that

$$\begin{aligned} c^1 &= \eta c \\ c^2 &= (1-\eta)c \end{aligned}$$

where $\eta \in [0, 1]$ is a function of ϕ_1 and γ.

Consequently, we can write the planner's first-order condition for k as

$$1 = \beta\alpha A k^{\alpha-1} \int \left(\frac{w_1(\epsilon) + Ak^\alpha e^\epsilon}{w_0 - k}\right)^{-\gamma} e^\epsilon g(\epsilon) d\epsilon$$

which is one equation to be solved for $k \geq 0$.

Anticipating a `Big K, little k` idea widely used in macroeconomics, to be discussed in detail below, let K be the value of k that solves the preceding equation so that

$$1 = \beta\alpha A K^{\alpha-1} \int \left(\frac{w_1(\epsilon) + AK^\alpha e^\epsilon}{w_0 - K}\right)^{-\gamma} g(\epsilon) e^\epsilon d\epsilon \tag{1}$$

The associated optimal consumption allocation is

$$\begin{aligned} C_0 &= w_0 - K \\ C_1(\epsilon) &= w_1(\epsilon) + AK^\alpha e^\epsilon \\ c_0^1 &= \eta C_0 \\ c_0^2 &= (1-\eta) C_0 \\ c_1^1(\epsilon) &= \eta C_1(\epsilon) \\ c_1^2(\epsilon) &= (1-\eta) C_1(\epsilon) \end{aligned}$$

where $\eta \in [0, 1]$ is the consumption share parameter mentioned above that is a function of the Pareto weight ϕ_1 and the utility curvature parameter γ.

Remarks

The relative Pareto weight parameter η does not appear in equation (1) that determines K.

Neither does it influence C_0 or $C_1(\epsilon)$, which depend solely on K.

The role of η is to determine how to allocate total consumption between the two types of consumers.

Thus, the planner's choice of K does not interact with how it wants to allocate consumption.

36.2 Competitive equilibrium

We now describe a competitive equilibrium for an economy that has specifications of consumer preferences, technology, and aggregate endowments that are identical to those in the preceding planning problem.

While prices do not appear in the planning problem – only quantities do – prices play an important role in a competitive equilibrium.

To understand how the planning economy is related to a competitive equilibrium, we now turn to the `Big K, little k` distinction.

36.2.1 Measures of agents and firms

We follow BCG in assuming that there are unit measures of

- consumers of type $i = 1$
- consumers of type $i = 2$
- firms with access to the production technology that converts k units of time 0 good into $Ak^\alpha e^\epsilon$ units of the time 1 good in random state ϵ

Thus, let $\omega \in [0, 1]$ index a particular consumer of type i.

Then define Big C^i as

$$C^i = \int_0^1 c^i(\omega) d\,\omega$$

In the same spirit, let $\zeta \in [0, 1]$ index a particular firm. Then define Big K as

$$K = \int_0^1 k(\zeta) d\,\zeta$$

The assumption that there are continua of our three types of agents plays an important role making each individual agent into a powerless **price taker**:

- an individual consumer chooses its own (infinesimal) part $c^i(\omega)$ of C^i taking prices as given
- an individual firm chooses its own (infinitesmimal) part $k(\zeta)$ of K taking prices as
- equilibrium prices depend on the `Big K, Big C` objects K and C

Nevertheless, in equilibrium, $K = k, C^i = c^i$

The assumption about measures of agents is thus a powerful device for making a host of competitive agents take as given equilibrium prices that are determined by the independent decisions of hosts of agents who behave just like they do.

Ownership

Consumers of type i own the following exogenous quantities of the consumption good in periods 0 and 1:

$$\begin{aligned} w_0^i, \quad i &= 1,2 \\ w_1^i(\epsilon) \quad i &= 1,2 \end{aligned}$$

where

$$\begin{aligned} \sum_i w_0^i &= w_0 \\ \sum_i w_1^i(\epsilon) &= w_1(\epsilon) \end{aligned}$$

Consumers also own shares in a firm that operates the technology for converting nonnegative amounts of the time 0 consumption good one-for-one into a capital good k that produces $Ak^\alpha e^\epsilon$ units of the time 1 consumption good in time 1 state ϵ.

Consumers of types $i = 1,2$ are endowed with θ_0^i shares of a firm and

$$\theta_0^1 + \theta_0^2 = 1$$

Asset markets

At time 0, consumers trade the following assets with other consumers and with firms:

- equities (also known as stocks) issued by firms
- one-period Arrow securities that pay one unit of consumption at time 1 when the shock ϵ assumes a particular value

Later, we'll allow the firm to issue bonds too, but not now.

36.2.2 Objects appearing in a competitive equilibrium

Let

- $a^i(\epsilon)$ be consumer i 's purchases of claims on time 1 consumption in state ϵ
- $q(\epsilon)$ be a pricing kernel for one-period Arrow securities
- $\theta_0^i \geq 0$ be consumer i's intial share of the firm, $\sum_i \theta_0^i = 1$
- θ^i be the fraction of a firm's shares purchased by consumer i at time $t = 0$
- V be the value of the representative firm
- $\tilde{V}$ be the value of equity issued by the representative firm
- K, C_0 be two scalars and $C_1(\epsilon)$ a function that we use to construct a guess about an equilibrium pricing kernel for Arrow securities

We proceed to describe constrained optimum problems faced by consumers and a representative firm in a competitive equilibrium.

36.2.3 A representative firm's problem

A representative firm takes Arrow security prices $q(\epsilon)$ as given.

The firm purchases capital $k \geq 0$ from consumers at time 0 and finances itself by issuing equity at time 0.

The firm produces time 1 goods $Ak^{\alpha}e^{\epsilon}$ in state ϵ and pays all of these `earnings` to owners of its equity.

The value of a firm's equity at time 0 can be computed by multiplying its state-contingent earnings by their Arrow securities prices and then adding over all contingencies:

$$\tilde{V} = \int Ak^{\alpha}e^{\epsilon}q(\epsilon)d\epsilon$$

Owners of a firm want it to choose k to maximize

$$V = -k + \int Ak^{\alpha}e^{\epsilon}q(\epsilon)d\epsilon$$

The firm's first-order necessary condition for an optimal k is

$$-1 + \alpha Ak^{\alpha-1}\int e^{\epsilon}q(\epsilon)d\epsilon = 0$$

The time 0 value of a representative firm is

$$V = -k + \tilde{V}$$

The right side equals the value of equity minus the cost of the time 0 goods that it purchases and uses as capital.

36.2.4 A consumer's problem

We now pose a consumer's problem in a competitive equilibrium.

As a price taker, each consumer faces a given Arrow securities pricing kernel $q(\epsilon)$, a given value of a firm V that has chosen capital stock k, a price of equity $\tilde{V}$, and prospective next period random dividends $Ak^{\alpha}e^{\epsilon}$.

If we evaluate consumer i's time 1 budget constraint at zero consumption $c_1^i(\epsilon) = 0$ and solve for $-a^i(\epsilon)$ we obtain

$$-\bar{a}^i(\epsilon;\theta^i) = w_1^i(\epsilon) + \theta^i Ak^{\alpha}e^{\epsilon} \tag{2}$$

The quantity $-\bar{a}^i(\epsilon;\theta^i)$ is the maximum amount that it is feasible for consumer i to repay to his Arrow security creditors at time 1 in state ϵ.

Notice that $-\bar{a}^i(\epsilon;\theta^i)$ defined in (2) depends on

- his endowment $w_1^i(\epsilon)$ at time 1 in state ϵ
- his share θ^i of a representive firm's dividends

These constitute two sources of **collateral** that back the consumer's issues of Arrow securities that pay off in state ϵ

Consumer i chooses a scalar c_0^i and a function $c_1^i(\epsilon)$ to maximize

$$u(c_0^i) + \beta\int u(c_1^i(\epsilon))g(\epsilon)d\epsilon$$

subject to time 0 and time 1 budget constraints

$$c_0^i \leq w_0^i + \theta_0^i V - \int q(\epsilon)a^i(\epsilon)d\epsilon - \theta^i\tilde{V}$$
$$c_1^i(\epsilon) \leq w_1^i(\epsilon) + \theta^i Ak^{\alpha}e^{\epsilon} + a^i(\epsilon)$$

Attach Lagrange multiplier λ_0^i to the budget constraint at time 0 and scaled Lagrange multiplier $\beta\lambda_1^i(\epsilon)g(\epsilon)$ to the budget constraint at time 1 and state ϵ, then form the Lagrangian

$$\begin{aligned} L^i = & u(c_0^i) + \beta \int u(c_1^i(\epsilon))g(\epsilon)d\epsilon \\ & + \lambda_0^i[w_0^i + \theta_0^i - \int q(\epsilon)a^i(\epsilon)d\epsilon - \theta^i \tilde{V} - c_0^i] \\ & + \beta \int \lambda_1^i(\epsilon)[w_1^i(\epsilon) + \theta^i A k^\alpha e^\epsilon + a^i(\epsilon) c_1^i(\epsilon)]g(\epsilon)d\epsilon \end{aligned}$$

Off corners, first-order necessary conditions for an optimum with respect to c_0^i, $c_1^i(\epsilon)$, and $a^i(\epsilon)$ are

$$\begin{aligned} c_0^i: & \quad u'(c_0^i) - \lambda_0^i = 0 \\ c_1^i(\epsilon): & \quad \beta u'(c_1^i(\epsilon))g(\epsilon) - \beta\lambda_1^i(\epsilon)g(\epsilon) = 0 \\ a^i(\epsilon): & \quad -\lambda_0^i q(\epsilon) + \beta\lambda_1^i(\epsilon) = 0 \end{aligned}$$

These equations imply that consumer i adjusts its consumption plan to satisfy

$$q(\epsilon) = \beta \left(\frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} \right) g(\epsilon) \tag{3}$$

To deduce a restriction on equilibrium prices, we solve the period 1 budget constraint to express $a^i(\epsilon)$ as

$$a^i(\epsilon) = c_1^i(\epsilon) - w_1^i(\epsilon) - \theta^i A k^\alpha e^\epsilon$$

then substitute the expression on the right side into the time 0 budget constraint and rearrange to get the single intertemporal budget constraint

$$w_0^i + \theta_0^i V + \int w_1^i(\epsilon)q(\epsilon)d\epsilon + \theta^i \left[A k^\alpha \int e^\epsilon q(\epsilon)d\epsilon - \tilde{V} \right] \geq c_0^i + \int c_1^i(\epsilon)q(\epsilon)d\epsilon \tag{4}$$

The right side of inequality (4) is the present value of consumer i's consumption while the left side is the present value of consumer i's endowment when consumer i buys θ^i shares of equity.

From inequality (4), we deduce two findings.

1. No arbitrage profits condition:

Unless

$$\tilde{V} = A k^\alpha \int e^\epsilon q(\epsilon)d\epsilon \tag{5}$$

an **arbitrage** opportunity would be open.

If

$$\tilde{V} > A k^\alpha \int e^\epsilon q(\epsilon)d\epsilon$$

the consumer could afford an arbitrarily high present value of consumption by setting θ^i to an arbitrarily large **negative** number.

If

$$\tilde{V} < A k^\alpha \int e^\epsilon q(\epsilon)d\epsilon$$

the consumer could afford an arbitrarily high present value of consumption by setting θ^i to be arbitrarily large **positive** number.

Since resources are finite, there can exist no such arbitrage opportunity in a competitive equilibrium.

Therefore, it must be true that the following no arbitrage condition prevails:

$$\tilde{V} = \int Ak^{\alpha} e^{\epsilon} q(\epsilon; K) d\epsilon \tag{6}$$

Equation (6) asserts that the value of equity equals the value of the state-contingent dividends $Ak^{\alpha} e^{\epsilon}$ evaluated at the Arrow security prices $q(\epsilon; K)$ that we have expressed as a function of K.

We'll say more about this equation later.

2. Indeterminacy of portfolio

When the no-arbitrage pricing equation (6) prevails, a consumer of type i's choice θ^i of equity is indeterminate.

Consumer of type i can offset any choice of θ^i by setting an appropriate schedule $a^i(\epsilon)$ for purchasing state-contingent securities.

36.2.5 Computing competitive equilibrium prices and quantities

Having computed an allocation that solves the planning problem, we can readily compute a competitive equilibrium via the following steps that, as we'll see, relies heavily on the `Big K, little k`, `Big C, little c` logic mentioned earlier:

- a competitive equilbrium allocation equals the allocation chosen by the planner
- competitive equilibrium prices and the value of a firm's equity are encoded in shadow prices from the planning problem that depend on Big K and Big C.

To substantiate that this procedure is valid, we proceed as follows.

With K in hand, we make the following guess for competitive equilibrium Arrow securities prices

$$q(\epsilon; K) = \beta \left(\frac{u'\left(w_1(\epsilon) + AK^{\alpha} e^{\epsilon}\right)}{u'(w_0 - K)} \right)^{-\gamma} \tag{7}$$

To confirm the guess, we begin by considering its consequences for the firm's choice of k.

With Arrow securities prices (7), the firm's first-order necessary condition for choosing k becomes

$$-1 + \alpha A k^{\alpha - 1} \int e^{\epsilon} q(\epsilon; K) d\epsilon = 0 \tag{8}$$

which can be verified to be satisfied if the firm sets

$$k = K$$

because by setting $k = K$ equation (8) becomes equivalent with the planner's first-order condition (1) for setting K.

To pose a consumer's problem in a competitive equilibrium, we require not only the above guess for the Arrow securities pricing kernel $q(\epsilon)$ but the value of equity $\tilde{V}$:

$$\tilde{V} = \int AK^{\alpha} e^{\epsilon} q(\epsilon; K) d\epsilon \tag{9}$$

Let $\tilde{V}$ be the value of equity implied by Arrow securities price function (7) and formula (9).

At the Arrow securities prices $q(\epsilon)$ given by (7) and equity value $\tilde{V}$ given by (9), consumer $i = 1, 2$ choose consumption allocations and portolios that satisfy the first-order necessary conditions

$$\beta \left(\frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} \right) g(\epsilon) = q(\epsilon; K)$$

It can be verified directly that the following choices satisfy these equations

$$
\begin{aligned}
c_0^1 + c_0^2 &= C_0 = w_0 - K \\
c_0^1(\epsilon) + c_0^2(\epsilon) &= C_1(\epsilon) = w_1(\epsilon) + Ak^\alpha e^\epsilon \\
\frac{c_1^2(\epsilon)}{c_1^1(\epsilon)} &= \frac{c_0^2}{c_0^1} = \frac{1-\eta}{\eta}
\end{aligned}
$$

for an $\eta \in (0,1)$ that depends on consumers' endowments $[w_0^1, w_0^2, w_1^1(\epsilon), w_1^2(\epsilon), \theta_0^1, \theta_0^2]$.

Remark: Multiple arrangements of endowments $[w_0^1, w_0^2, w_1^1(\epsilon), w_1^2(\epsilon), \theta_0^1, \theta_0^2]$ associated with the same distribution of wealth η. Can you explain why? **Hint:** Think about the portfolio indeterminacy finding above.

36.2.6 Modigliani-Miller theorem

We now allow a firm to issue both bonds and equity.

Payouts from equity and bonds, respectively, are

$$
\begin{aligned}
d^e(k, b; \epsilon) &= \max\left\{e^\epsilon Ak^\alpha - b, 0\right\} \\
d^b(k, b; \epsilon) &= \min\left\{\frac{e^\epsilon Ak^\alpha}{b}, 1\right\}
\end{aligned}
$$

Thus, one unit of the bond pays one unit of consumption at time 1 in state ϵ if $Ak^\alpha e^\epsilon - b \geq 0$, which is true when $\epsilon \geq \epsilon^* = \log \frac{b}{Ak^\alpha}$, and pays $\frac{Ak^\alpha e^\epsilon}{b}$ units of time 1 consumption in state ϵ when $\epsilon < \epsilon^*$.

The value of the firm is now the sum of equity plus the value of bonds, which we denote

$$
\tilde{V} + bp(k, b)
$$

where $p(k, b)$ is the price of one unit of the bond when a firm with k units of physical capital issues b bonds.

We continue to assume that there are complete markets in Arrow securities with pricing kernel $q(\epsilon)$.

A version of the no-arbitrage-in-equilibrium argument that we presented earlier implies that the value of equity and the price of bonds are

$$
\begin{aligned}
\tilde{V} &= Ak^\alpha \int_{\epsilon^*}^{\infty} e^\epsilon q(\epsilon) d\epsilon - b \int_{\epsilon^*}^{\infty} q(\epsilon) d\epsilon \\
p(k, b) &= \frac{Ak^\alpha}{b} \int_{-\infty}^{\epsilon^*} e^\epsilon q(\epsilon) d\epsilon + \int_{\epsilon^*}^{\infty} q(\epsilon) d\epsilon
\end{aligned}
$$

Consequently, the value of the firm is

$$
\tilde{V} + p(k, b) b = Ak^\alpha \int_{-\infty}^{\infty} e^\epsilon q(\epsilon) d\epsilon,
$$

which is the same expression that we obtained above when we assumed that the firm issued only equity.

We thus obtain a version of the celebrated Modigliani-Miller theorem [MM58] about firms' finance:

Modigliani-Miller theorem:

- The value of a firm is independent the mix of equity and bonds that it uses to finance its physical capital.
- The firms's decision about how much physical capital to purchase does not depend on whether it finances those purchases by issuing bonds or equity
- The firm's choice of whether to finance itself by issuing equity or bonds is indeterminant

Please note the role of the assumption of complete markets in Arrow securities in substantiating these claims.

In *Equilibrium Capital Structures with Incomplete Markets*, we will assume that markets are (very) incomplete – we'll shut down markets in almost all Arrow securities.

That will pull the rug from underneath the Modigliani-Miller theorem.

36.3 Code

We create a class object `BCG_complete_markets` to compute equilibrium allocations of the complete market BCG model given a list of parameter values.

It consists of 4 functions that do the following things:

- `opt_k` computes the planner's optimal capital K
  - First, create a grid for capital.
  - Then for each value of capital stock in the grid, compute the left side of the planner's first-order necessary condition for k, that is,

$$\beta \alpha A K^{\alpha-1} \int \left(\frac{w_1(\epsilon) + A K^\alpha e^\epsilon}{w_0 - K} \right)^{-\gamma} e^\epsilon g(\epsilon) d\epsilon - 1 = 0$$

  - Find k that solves this equation.
- `q` computes Arrow security prices as a function of the productivity shock ϵ and capital K:

$$q(\epsilon; K) = \beta \left(\frac{u'\left(w_1(\epsilon) + A K^\alpha e^\epsilon\right)}{u'(w_0 - K)} \right)$$

- `V` solves for the firm value given capital k:

$$V = -k + \int A k^\alpha e^\epsilon q(\epsilon; K) d\epsilon$$

- `opt_c` computes optimal consumptions c^i_0, and $c^i(\epsilon)$:
  - The function first computes weight η using the budget constraint for agent 1:

$$w^1_0 + \theta^1_0 V + \int w^1_1(\epsilon) q(\epsilon) d\epsilon = c^1_0 + \int c^1_1(\epsilon) q(\epsilon) d\epsilon = \eta \left(C_0 + \int C_1(\epsilon) q(\epsilon) d\epsilon \right)$$

    where

$$\begin{aligned} C_0 &= w_0 - K \\ C_1(\epsilon) &= w_1(\epsilon) + A K^\alpha e^\epsilon \end{aligned}$$

  - It computes consumption for each agent as

$$\begin{aligned} c^1_0 &= \eta C_0 \\ c^2_0 &= (1-\eta) C_0 \\ c^1_1(\epsilon) &= \eta C_1(\epsilon) \\ c^2_1(\epsilon) &= (1-\eta) C_1(\epsilon) \end{aligned}$$

The list of parameters includes:

- χ_1, χ_2: Correlation parameters for agents 1 and 2. Default values are 0 and 0.9, respectively.
- w^1_0, w^2_0: Initial endowments. Default values are 1.
- θ^1_0, θ^2_0: Consumers' initial shares of a representative firm. Default values are 0.5.
- ψ: CRRA risk parameter. Default value is 3.
- α: Returns to scale production function parameter. Default value is 0.6.
- A: Productivity of technology. Default value is 2.5.
- μ, σ: Mean and standard deviation of the log of the shock. Default values are -0.025 and 0.4, respectively.
- β: time preference discount factor. Default value is .96.
- `nb_points_integ`: number of points used for integration through Gauss-Hermite quadrature: default value is 10

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from numba import njit, prange
from quantecon.optimize import root_finding
%matplotlib inline
```

```python
#=========== Class: BCG for complete markets ===========#
class BCG_complete_markets:

    # init method or constructor
    def __init__(self,
                 □1 = 0,
                 □2 = 0.9,
                 w10 = 1,
                 w20 = 1,
                 □10 = 0.5,
                 □20 = 0.5,
                 □ = 3,
                 □ = 0.6,
                 A = 2.5,
                 □ = -0.025,
                 □ = 0.4,
                 □ = 0.96,
                 nb_points_integ = 10):

        #=========== Setup ===========#
        # Risk parameters
        self.□1 = □1
        self.□2 = □2

        # Other parameters
        self.□ = □
        self.□ = □
        self.A = A
        self.□ = □
        self.□ = □
        self.□ = □

        # Utility
```

(continues on next page)

(continued from previous page)

```
        self.u = lambda c: (c**(1-□)) / (1-□)

        # Production
        self.f = njit(lambda k: A * (k ** □))
        self.Y = lambda □, k: np.exp(□) * self.f(k)

        # Initial endowments
        self.w10 = w10
        self.w20 = w20
        self.w0 = w10 + w20

        # Initial holdings
        self.□10 = □10
        self.□20 = □20

        # Endowments at t=1
        w11 = njit(lambda □: np.exp(-□1*□ - 0.5*(□1**2)*(□**2) + □1*□))
        w21 = njit(lambda □: np.exp(-□2*□ - 0.5*(□2**2)*(□**2) + □2*□))
        self.w11 = w11
        self.w21 = w21

        self.w1 = njit(lambda □: w11(□) + w21(□))

        # Normal PDF
        self.g = lambda x: norm.pdf(x, loc=□, scale=□)

        # Integration
        x, self.weights = np.polynomial.hermite.hermgauss(nb_points_integ)
        self.points_integral = np.sqrt(2) * □ * x + □

        self.k_foc = k_foc_factory(self)

    #=========== Optimal k ===========#
    # Function: solve for optimal k
    def opt_k(self, plot=False):
        w0 = self.w0

        # Grid for k
        kgrid = np.linspace(1e-4, w0-1e-4, 100)

        # get FONC values for each k in the grid
        kfoc_list = [];
        for k in kgrid:
            kfoc = self.k_foc(k, self.□1, self.□2)
            kfoc_list.append(kfoc)

        # Plot FONC for k
        if plot:
            fig, ax = plt.subplots(figsize=(8,7))
            ax.plot(kgrid, kfoc_list, color='blue', label=r'FONC for k')
            ax.axhline(0, color='red', linestyle='--')
            ax.legend()
            ax.set_xlabel(r'k')
            plt.show()

        # Find k that solves the FONC
        kk = root_finding.newton_secant(self.k_foc, 1e-2, args=(self.□1, self.□2)).
↪root
```

(continues on next page)

(continued from previous page)

```
        return kk

    #=========== Arrow security price ===========#
    # Function: Compute Arrow security price
    def q(self,□,k):
        □ = self.□
        □ = self.□
        w0 = self.w0
        w1 = self.w1
        fk = self.f(k)
        g = self.g

        return □ * ((w1(□) + np.exp(□)*fk) / (w0 - k))**(-□)


    #=========== Firm value V ===========#
    # Function: compute firm value V
    def V(self, k):
        q = self.q
        fk = self.f(k)
        weights = self.weights
        integ = lambda □: np.exp(□) * fk * q(□, k)

        return -k + np.sum(weights * integ(self.points_integral)) / np.sqrt(np.pi)

    #=========== Optimal c ===========#
    # Function: Compute optimal consumption choices c
    def opt_c(self, k=None, plot=False):
        w1 = self.w1
        w0 = self.w0
        w10 = self.w10
        w11 = self.w11
        □10 = self.□10
        Y = self.Y
        q = self.q
        V = self.V
        weights = self.weights

        if k is None:
            k = self.opt_k()

        # Solve for the ratio of consumption □ from the intertemporal B.C.
        fk = self.f(k)

        c1 = lambda □: (w1(□) + np.exp(□)*fk)*q(□,k)
        denom = np.sum(weights * c1(self.points_integral)) / np.sqrt(np.pi) + (w0 - k)

        w11q = lambda □: w11(□)*q(□,k)
        num = w10 + □10 * V(k) + np.sum(weights * w11q(self.points_integral)) / np.
↪sqrt(np.pi)

        □ = num / denom

        # Consumption choices
        c10 = □ * (w0 - k)
        c20 = (1-□) * (w0 - k)
```

(continues on next page)

(continued from previous page)

```
        c11 = lambda □: □ * (w1(□)+Y(□,k))
        c21 = lambda □: (1-□) * (w1(□)+Y(□,k))

        return c10, c20, c11, c21


def k_foc_factory(model):
    □ = model.□
    f = model.f
    □ = model.□
    □ = model.□
    A = model.A
    □ = model.□
    w0 = model.w0
    □ = model.□
    □ = model.□

    weights = model.weights
    points_integral = model.points_integral

    w11 = njit(lambda □, □1, : np.exp(-□1*□ - 0.5*(□1**2)*(□**2) + □1*□))
    w21 = njit(lambda □, □2: np.exp(-□2*□ - 0.5*(□2**2)*(□**2) + □2*□))
    w1 = njit(lambda □, □1, □2: w11(□, □1) + w21(□, □2))

    @njit
    def integrand(□, □1, □2, k=1e-4):
        fk = f(k)
        return (w1(□, □1, □2) + np.exp(□) * fk) ** (-□) * np.exp(□)

    @njit
    def k_foc(k, □1, □2):
        int_k = np.sum(weights * integrand(points_integral, □1, □2, k=k)) / np.
↪sqrt(np.pi)

        mul = □ * □ * A * k ** (□ - 1) / ((w0 - k) ** (-□))
        val = mul * int_k - 1

        return val

    return k_foc
```

36.3.1 Examples

Below we provide some examples of how to use `BCG_complete markets`.

1st example

In the first example, we set up instances of BCG complete markets models.

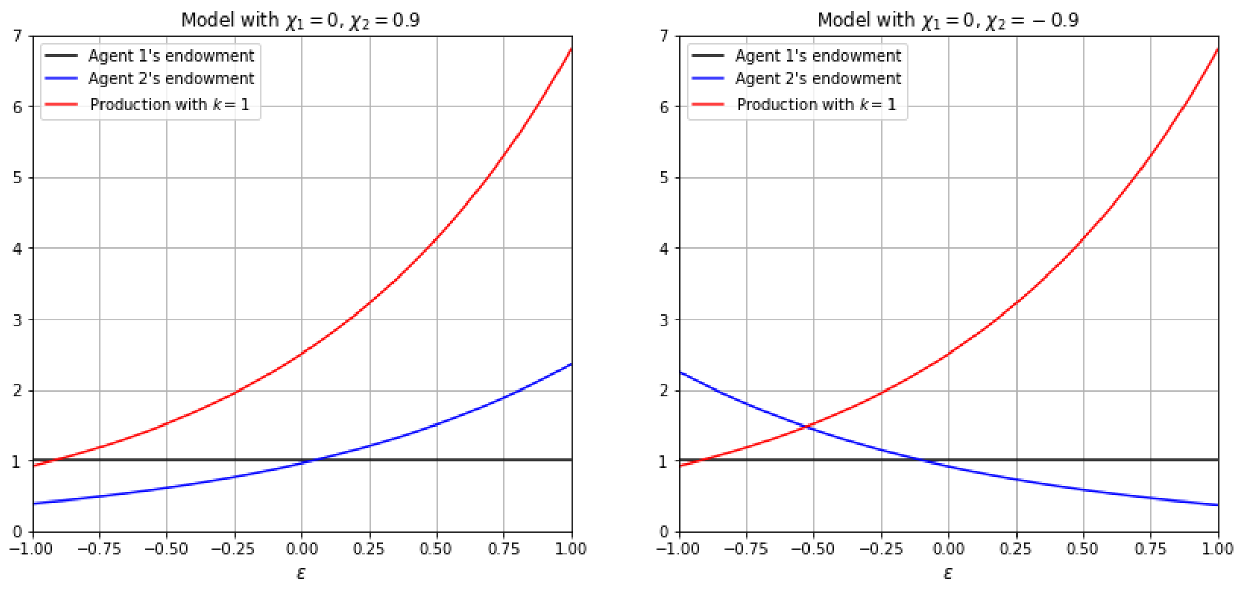
We can use either default parameter values or set parameter values as we want.

The two instances of the BCG complete markets model, `mdl1` and `mdl2`, represent the model with default parameter settings and with agent 2's income correlation altered to be $\chi_2 = -0.9$, respectively.

```
# Example: BCG model for complete markets
mdl1 = BCG_complete_markets()
mdl2 = BCG_complete_markets(□2=-0.9)
```

Let's plot the agents' time-1 endowments with respect to shocks to see the difference in the two models:

```
#==== Figure 1: HH endowments and firm productivity ====#
# Realizations of innovation from -3 to 3
epsgrid = np.linspace(-1,1,1000)


fig, ax = plt.subplots(1,2,figsize=(14,6))
ax[0].plot(epsgrid, mdl1.w11(epsgrid), color='black', label='Agent 1\'s endowment')
ax[0].plot(epsgrid, mdl1.w21(epsgrid), color='blue', label='Agent 2\'s endowment')
ax[0].plot(epsgrid, mdl1.Y(epsgrid,1), color='red', label=r'Production with $k=1$')
ax[0].set_xlim([-1,1])
ax[0].set_ylim([0,7])
ax[0].set_xlabel(r'$\epsilon$',fontsize=12)
ax[0].set_title(r'Model with $\chi_1 = 0$, $\chi_2 = 0.9$')
ax[0].legend()
ax[0].grid()

ax[1].plot(epsgrid, mdl2.w11(epsgrid), color='black', label='Agent 1\'s endowment')
ax[1].plot(epsgrid, mdl2.w21(epsgrid), color='blue', label='Agent 2\'s endowment')
ax[1].plot(epsgrid, mdl2.Y(epsgrid,1), color='red', label=r'Production with $k=1$')
ax[1].set_xlim([-1,1])
ax[1].set_ylim([0,7])
ax[1].set_xlabel(r'$\epsilon$',fontsize=12)
ax[1].set_title(r'Model with $\chi_1 = 0$, $\chi_2 = -0.9$')
ax[1].legend()
ax[1].grid()

plt.show()
```



Let's also compare the optimal capital stock, k, and optimal time-0 consumption of agent 2, c_0^2, for the two models:

```
# Print optimal k
kk_1 = mdl1.opt_k()
kk_2 = mdl2.opt_k()

print('The optimal k for model 1: {:.5f}'.format(kk_1))
print('The optimal k for model 2: {:.5f}'.format(kk_2))

# Print optimal time-0 consumption for agent 2
c20_1 = mdl1.opt_c(k=kk_1)[1]
c20_2 = mdl2.opt_c(k=kk_2)[1]

print('The optimal c20 for model 1: {:.5f}'.format(c20_1))
print('The optimal c20 for model 2: {:.5f}'.format(c20_2))
```

```
The optimal k for model 1: 0.14235
The optimal k for model 2: 0.13791
```

```
The optimal c20 for model 1: 0.90205
The optimal c20 for model 2: 0.92862
```

2nd example

In the second example, we illustrate how the optimal choice of k is influenced by the correlation parameter χ_i.

We will need to install the `plotly` package for 3D illustration. See https://plotly.com/python/getting-started/ for further instructions.

```
# Mesh grid of 
N = 30
1grid, 2grid = np.meshgrid(np.linspace(-1,1,N),
                             np.linspace(-1,1,N))

k_foc = k_foc_factory(mdl1)
```

(continues on next page)

(continued from previous page)

```
# Create grid for k
kgrid = np.zeros_like(□1grid)

w0 = mdl1.w0

@njit(parallel=True)
def fill_k_grid(kgrid):
    # Loop: Compute optimal k and
    for i in prange(N):
        for j in prange(N):
            X1 = □1grid[i, j]
            X2 = □2grid[i, j]
            k = root_finding.newton_secant(k_foc, 1e-2, args=(X1, X2)).root
            kgrid[i, j] = k
```

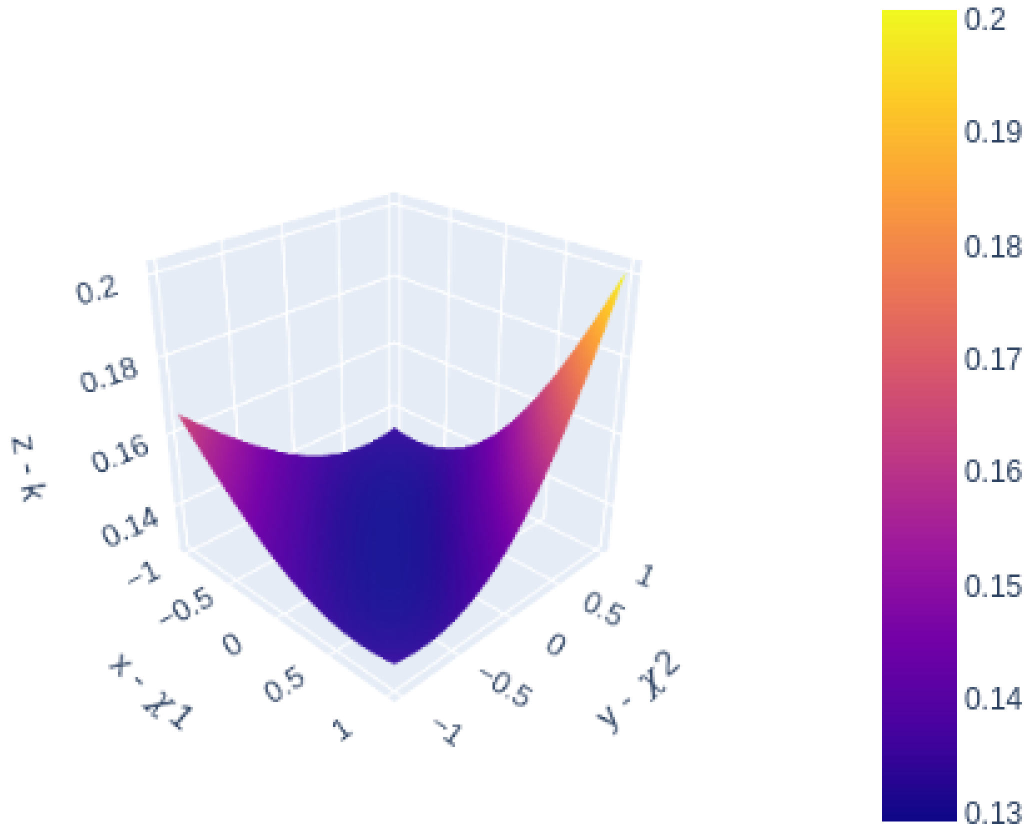
```
%%time
fill_k_grid(kgrid)
```

```
CPU times: user 3.02 s, sys: 7.65 ms, total: 3.03 s
Wall time: 3.01 s
```

```
%%time
# Second-run
fill_k_grid(kgrid)
```

```
CPU times: user 10.1 ms, sys: 12 µs, total: 10.1 ms
Wall time: 9.66 ms
```

```
#=== Example: Plot optimal k with different correlations ===#

from IPython.display import Image
# Import plotly
import plotly.graph_objs as go

# Plot optimal k
fig = go.Figure(data=[go.Surface(x=□1grid, y=□2grid, z=kgrid)])
fig.update_layout(scene = dict(xaxis_title='x - □1',
                               yaxis_title='y - □2',
                               zaxis_title='z - k',
                               aspectratio=dict(x=1,y=1,z=1)))
fig.update_layout(width=500,
                  height=500,
                  margin=dict(l=50, r=50, b=65, t=90))
fig.update_layout(scene_camera=dict(eye=dict(x=2, y=-2, z=1.5)))

# Export to PNG file
Image(fig.to_image(format="png"))
# fig.show() will provide interactive plot when running
# notebook locally
```


0.2
0.19
0.18
0.17
0.16
0.15
0.14
0.13
0.2
0.18
0.16
0.14
z - k
x - χ1
y - χ2
−1
−0.5
0
0.5
1

CHAPTER

THIRTYSEVEN

EQUILIBRIUM CAPITAL STRUCTURES WITH INCOMPLETE MARKETS

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
!pip install interpolation
!conda install -y -c plotly plotly plotly-orca
```

37.1 Introduction

This is an extension of an earlier lecture *Irrelevance of Capital Structure with Complete Markets* about a **complete markets** model.

In contrast to that lecture, this one describes an instance of a model authored by Bisin, Clementi, and Gottardi [BCG18] in which financial markets are **incomplete**.

Instead of being able to trade equities and a full set of one-period Arrow securities as they can in *Irrelevance of Capital Structure with Complete Markets*, here consumers and firms trade only equity and a bond.

It is useful to watch how outcomes differ in the two settings.

In the complete markets economy in *Irrelevance of Capital Structure with Complete Markets*

- there is a unique stochastic discount factor that prices all assets
- consumers' portfolio choices are indeterminate

- firms' financial structures are indeterminate, so the model embodies an instance of a Modigliani-Miller irrelevance theorem [MM58]
- the aggregate of all firms' financial structures are indeterminate, a consequence of there being redundant assets

In the incomplete markets economy studied here

- there is a not a unique equilibrium stochastic discount factor
- different stochastic discount factors price different assets
- consumers' portfolio choices are determinate
- while **individual** firms' financial structures are indeterminate, thus conforming to part of a Modigliani-Miller theorem, [MM58], the **aggregate** of all firms' financial structures **is** determinate.

A `Big K, little k` analysis played an important role in the previous lecture *Irrelevance of Capital Structure with Complete Markets*.

A more subtle version of a `Big K, little k` features in the BCG incomplete markets environment here.

We use it to convey the heart of what BCG call a **rational conjectures** equilibrium in which conjectures are about equilibrium pricing functions in regions of the state space that an average consumer or firm does not visit in equilibrium.

Note that the absence of complete markets means that we can compute competitive equilibrium prices and allocations by first solving the simple planning problem that we did in *Irrelevance of Capital Structure with Complete Markets*.

Instead, we compute an equilibrium by solving a system of simultaneous inequalities.

(Here we do not address the interesting question of whether there is a *different* planning problem that we could use to compute a competitive equlibrium allocation.)

37.1.1 Setup

We adopt specifications of preferences and technologies used by Bisin, Clemente, and Gottardi (2018) [BCG18] and in our earlier lecture on a complete markets version of their model.

The economy lasts for two periods, $t = 0, 1$.

There are two types of consumers named $i = 1, 2$.

A scalar random variable ϵ affects both

- a representative firm's physical return $f(k)e^\epsilon$ in period 1 from investing $k \geq 0$ in capital in period 0.
- period 1 endowments $w_1^i(\epsilon)$ of the consumption good for agents $i = 1$ and $i = 2$.

37.1.2 Ownership

A consumer of type i is endowed with w_0^i units of the time 0 good and $w_1^i(\epsilon)$ of the time 1 good when the random variable takes value ϵ.

At the start of period 0, a consumer of type i also owns θ_0^i shares of a representative firm.

37.1.3 Measures of agents and firms

As in the companion lecture *Irrelevance of Capital Structure with Complete Markets* that studies a complete markets version of the model, we follow BCG in assuming that there are unit measures of

- consumers of type $i = 1$
- consumers of type $i = 2$
- firms with access to a production technology that converts k units of time 0 good into $Ak^\alpha e^\epsilon$ units of the time 1 good in random state ϵ

Thus, let $\omega \in [0, 1]$ index a particular consumer of type i.

Then define Big C^i as

$$C^i = \int_0^1 c^i(\omega) d\,\omega$$

with components

$$C_0^i = \int_0^1 c_0^i(\omega) d\,\omega$$
$$C_1^i(\epsilon) = \int_0^1 c_1^i(\epsilon; \omega) d\,\omega$$

In the same spirit, let $\zeta \in [0, 1]$ index a particular firm and let firm ζ purchase $k(\zeta)$ units of capital and issue $b(\zeta)$ bonds.

Then define Big K and Big B as

$$K = \int_0^1 k(\zeta) d\,\zeta, \quad B = \int_0^1 b(\zeta) d\,\zeta$$

The assumption that there are equal measures of our three types of agents justifies our assumption that each individual agent is a powerless **price taker**:

- an individual consumer chooses its own (infinitesimal) part $c^i(\omega)$ of C^i taking prices as given
- an individual firm chooses its own (infinitesmimal) part $k(\zeta)$ of K and $b(\zeta)$ of B taking pricing functions as given
- However, equilibrium prices depend on the `Big K, Big B, Big C` objects K, B, and C

The assumption about measures of agents is a powerful device for making a host of competitive agents take as given the equilibrium prices that turn out to be determined by the decisions of hosts of agents who are just like them.

We call an equilibrium **symmetric** if

- all type i consumers choose the same consumption profiles so that $c^i(\omega) = C^i$ for all $\omega \in [0, 1]$
- all firms choose the same levels of k and b so that $k(\zeta) = K$, $b(\zeta) = B$ for all $\zeta \in [0, 1]$

In this lecture, we restrict ourselves to describing symmetric equilibria.

37.1.4 Endowments

Per capital economy-wide endowments in periods 0 and 1 are

$$
\begin{aligned}
w_0 &= w_0^1 + w_0^2 \\
w_1(\epsilon) &= w_1^1(\epsilon) + w_1^2(\epsilon) \text{ in state } \epsilon
\end{aligned}
$$

37.1.5 Feasibility:

Where $\alpha \in (0,1)$ and $A > 0$

$$
\begin{aligned}
C_0^1 + C_0^2 &= w_0^1 + w_0^2 - K \\
C_1^1(\epsilon) + C_1^2(\epsilon) &= w_1^1(\epsilon) + w_1^2(\epsilon) + e^{\epsilon} \int_0^1 f(k(\zeta))d\zeta, \quad k \geq 0
\end{aligned}
$$

where $f(k) = Ak^{\alpha}, A > 0, \alpha \in (0,1)$.

37.1.6 Parameterizations

Following BCG, we shall employ the following parameterizations:

$$
\begin{aligned}
\epsilon &\sim \mathcal{N}(\mu, \sigma^2) \\
u(c) &= \frac{c^{1-\gamma}}{1-\gamma} \\
w_1^i(\epsilon) &= e^{-\chi_i \mu - .5\chi_i^2 \sigma^2 + \chi_i \epsilon}, \quad \chi_i \in [0,1]
\end{aligned}
$$

Sometimes instead of asuming $\epsilon \sim g(\epsilon) = \mathcal{N}(0, \sigma^2)$, we'll assume that $g(\cdot)$ is a probability mass function that serves as a discrete approximation to a standardized normal density.

37.1.7 Preferences:

A consumer of type i orders period 0 consumption c_0^i and state ϵ-period 1 consumption $c^i(\epsilon)$ by

$$
u^i = u(c_0^i) + \beta \int u(c_1^i(\epsilon))g(\epsilon)d\epsilon, \quad i = 1, 2
$$

$\beta \in (0,1)$ and the one-period utility function is

$$
u(c) = \begin{cases} \frac{c^{1-\gamma}}{1-\gamma} & \text{if } \gamma \neq 1 \\ \log c & \text{if } \gamma = 1 \end{cases}
$$

37.1.8 Risk-sharing motives

The two types of agents' period 1 endowments have different correlations with the physical return on capital.

Endowment differences give agents incentives to trade risks that in the complete market version of the model showed up in their demands for equity and in their demands and supplies of one-period Arrow securities.

In the incomplete-markets setting under study here, these differences show up in differences in the two types of consumers' demands for a typical firm's bonds and equity, the only two assets that agents can now trade.

37.2 Asset Markets

Markets are incomplete: *ex cathedra* we the model builders declare that only equities and bonds issued by representative firms can be traded.

Let θ^i and ξ^i be a consumer of type i's post-trade holdings of equity and bonds, respectively.

A firm issues bonds promising to pay b units of consumption at time $t = 1$ and purchases k units of physical capital at time $t = 0$.

When $e^{\epsilon} A k^{\alpha} < b$ at time 1, the firm defaults and its output is divided equally among bondholders.

Evidently, when the productivity shock $\epsilon < \epsilon^* = \log\left(\frac{b}{Ak^{\alpha}}\right)$, the firm defaults on its debt

Payoffs to equity and debt at date 1 as functions of the productivity shock ϵ are thus

$$\begin{aligned} d^e(k, b; \epsilon) &= \max\left\{e^{\epsilon} A k^{\alpha} - b, 0\right\} \\ d^b(k, b; \epsilon) &= \min\left\{\frac{e^{\epsilon} A k^{\alpha}}{b}, 1\right\} \end{aligned} \tag{1}$$

A firm faces a bond price function $p(k, b)$ when it issues b bonds and purchases k units of physical capital.

A firm's equity is worth $q(k, b)$ when it issues b bonds and purchases k units of physical capital.

A firm regards an equity-pricing function $q(k, b)$ and a bond pricing function $p(k, b)$ as exogenous in the sense that they are not affected by its choices of k and b.

Consumers face equilibrium prices $\check{q}$ and $\check{p}$ for bonds and equities, where $\check{q}$ and $\check{p}$ are both scalars.

Consumers are price takers and only need to know the scalars $\check{q}, \check{p}$.

Firms are *price function* takers and must know the functions $q(k, b), p(k, b)$ in order completely to pose their optimum problems.

37.2.1 Consumers

Each consumer of type i is endowed with w_0^i of the time 0 consumption good, $w_1^i(\epsilon)$ of the time 1, state ϵ consumption good and also owns a fraction $\theta_0^i \in (0, 1)$ of the initial value of a representative firm, where $\theta_0^1 + \theta_0^2 = 1$.

The initial value of a representative firm is V (an object to be determined in a rational expectations equilibrium).

Consumer i buys θ^i shares of equity and buys bonds worth $\check{p}\xi^i$ where $\check{p}$ is the bond price.

Being a price-taker, a consumer takes V, $\check{q}$, $\check{p}$, and K, B as given.

Consumers know that equilibrium payoff functions for bonds and equities take the form

$$\begin{aligned} d^e(K, B; \epsilon) &= \max\left\{e^{\epsilon} A K^{\alpha} - B, 0\right\} \\ d^b(K, B; \epsilon) &= \min\left\{\frac{e^{\epsilon} A K^{\alpha}}{B}, 1\right\} \end{aligned}$$

Consumer i's optimization problem is

$$\begin{aligned} \max_{c_0^i, \theta^i, \xi^i, c_1^i(\epsilon)} \quad & u(c_0^i) + \beta \int u(c^i(\epsilon)) g(\epsilon) \, d\epsilon \\ \text{subject to} \quad & c_0^i = w_0^i + \theta_0^i V - \check{q}\theta^i - \check{p}\xi^i, \\ & c_1^i(\epsilon) = w_1^i(\epsilon) + \theta^i d^e(K, B; \epsilon) + \xi^i d^b(K, B; \epsilon) \ \forall \ \epsilon, \\ & \theta^i \geq 0, \xi^i \geq 0. \end{aligned}$$

The last two inequalities impose that the consumer cannot short sell either equity or bonds.

In a rational expectations equilibrium, $\check{q} = q(K, B)$ and $\check{p} = p(K, B)$

We form consumer i's Lagrangian:

$$\begin{aligned} L^i := & u(c_0^i) + \beta \int u(c^i(\epsilon)) g(\epsilon) \, d\epsilon \\ & + \lambda_0^i [w_0^i + \theta_0 V - \check{q}\theta^i - \check{p}\xi^i - c_0^i] \\ & + \beta \int \lambda_1^i(\epsilon) \left[w_1^i(\epsilon) + \theta^i d^e(K, B; \epsilon) + \xi^i d^b(K, B; \epsilon) - c_1^i(\epsilon) \right] g(\epsilon) \, d\epsilon \end{aligned}$$

Consumer i's first-order necessary conditions for an optimum include:

$$\begin{aligned} c_0^i : & \quad u'(c_0^i) = \lambda_0^i \\ c_1^i(\epsilon) : & \quad u'(c_1^i(\epsilon)) = \lambda_1^i(\epsilon) \\ \theta^i : & \quad \beta \int \lambda_1^i(\epsilon) d^e(K, B; \epsilon) g(\epsilon) \, d\epsilon \leq \lambda_0^i \check{q} \quad (= \text{ if } \theta^i > 0) \\ \xi^i : & \quad \beta \int \lambda_1^i(\epsilon) d^b(K, B; \epsilon) g(\epsilon) \, d\epsilon \leq \lambda_0^i \check{p} \quad (= \text{ if } b^i > 0) \end{aligned}$$

We can combine and rearrange consumer i's first-order conditions to become:

$$\begin{aligned} \check{q} & \geq \beta \int \frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} d^e(K, B; \epsilon) g(\epsilon) \, d\epsilon \quad (= \text{ if } \theta^i > 0) \\ \check{p} & \geq \beta \int \frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} d^b(K, B; \epsilon) g(\epsilon) \, d\epsilon \quad (= \text{ if } b^i > 0) \end{aligned}$$

These inequalities imply that in a symmetric rational expectations equilibrium consumption allocations and prices satisfy

$$\begin{aligned} \check{q} & = \max_i \beta \int \frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} d^e(K, B; \epsilon) g(\epsilon) \, d\epsilon \\ \check{p} & = \max_i \beta \int \frac{u'(c_1^i(\epsilon))}{u'(c_0^i)} d^b(K, B; \epsilon) g(\epsilon) \, d\epsilon \end{aligned}$$

37.2.2 Pricing functions

When individual firms solve their optimization problems, they take big C^i's as fixed objects that they don't influence.

A representative firm faces a price function $q(k, b)$ for its equity and a price function $p(k, b)$ per unit of bonds that satisfy

$$\begin{aligned} q(k, b) & = \max_i \beta \int \frac{u'(C_1^i(\epsilon))}{u'(C_0^i)} d^e(k, b; \epsilon) g(\epsilon) \, d\epsilon \\ p(k, b) & = \max_i \beta \int \frac{u'(C_1^i(\epsilon))}{u'(C_0^i)} d^b(k, b; \epsilon) g(\epsilon) \, d\epsilon \end{aligned}$$

where the payoff functions are described by equations (1).

Notice the appearance of big C^i's on the right sides of these two equations that define equilibrium pricing functions.

The two price functions describe outcomes not only for equilibrium choices K, B of capital k and debt b, but also for any **out-of-equilibrium** pairs $(k, b) \neq (K, B)$.

The firm is assumed to know both price functions.

This means that the firm understands that its choice of k, b influences how markets price its equity and debt.

This package of assumptions is sometimes called **rational conjectures** (about price functions).

BCG give credit to Makowski for emphasizing and clarifying how rational conjectures are components of rational expectations equilibria.

37.2.3 Firms

The firm chooses capital k and debt b to maximize its market value:

$$V \equiv \max_{k,b} -k + q(k,b) + p(k,b)b$$

Attributing value maximization to the firm is a good idea because in equilibrium consumers of both types *want* a firm to maximize its value.

In the special quantitative examples studied here

- consumers of types $i = 1, 2$ both hold equity
- only consumers of type $i = 2$ hold debt; consumers of type $i = 1$ hold none.

These outcomes occur because we follow BCG and set parameters so that a type 2 consumer's stochastic endowment of the consumption good in period 1 is more correlated with the firm's output than is a type 1 consumer's.

This gives consumers of type 2 a motive to hedge their second period endowment risk by holding bonds (they also choose to hold some equity).

These outcomes mean that the pricing functions end up satisfying

$$\begin{aligned} q(k,b) &= \beta \int \frac{u'(C_1^1(\epsilon))}{u'(C_0^1)} d^e(k,b;\epsilon) g(\epsilon)\ d\epsilon = \beta \int \frac{u'(C_1^2(\epsilon))}{u'(C_0^2)} d^e(k,b;\epsilon) g(\epsilon)\ d\epsilon \\ p(k,b) &= \beta \int \frac{u'(C_1^2(\epsilon))}{u'(C_0^2)} d^b(k,b;\epsilon) g(\epsilon)\ d\epsilon \end{aligned}$$

Recall that $\epsilon^*(k,b) \equiv \log\left(\frac{b}{Ak^\alpha}\right)$ is a firm's default threshold.

We can rewrite the pricing functions as:

$$\begin{aligned} q(k,b) &= \beta \int_{\epsilon^*}^{\infty} \frac{u'(C_1^i(\epsilon))}{u'(C_0^i)} \left(e^\epsilon Ak^\alpha - b\right) g(\epsilon)\ d\epsilon, \quad i = 1, 2 \\ p(k,b) &= \beta \int_{-\infty}^{\epsilon^*} \frac{u'(C_1^2(\epsilon))}{u'(C_0^2)} \left(\frac{e^\epsilon Ak^\alpha}{b}\right) g(\epsilon)\ d\epsilon + \beta \int_{\epsilon^*}^{\infty} \frac{u'(C_1^2(\epsilon))}{u'(C_0^2)} g(\epsilon)\ d\epsilon \end{aligned}$$

Firm's optimization problem

The firm's optimization problem is

$$V \equiv \max_{k,b} \left\{-k + q(k,b) + p(k,b)b\right\}$$

The firm's first-order necessary conditions with respect to k and b, respectively, are

$$\begin{aligned} k: &\quad -1 + \frac{\partial q(k,b)}{\partial k} + b\frac{\partial p(q,b)}{\partial k} = 0 \\ b: &\quad \frac{\partial q(k,b)}{\partial b} + p(k,b) + b\frac{\partial p(k,b)}{\partial b} = 0 \end{aligned}$$

We use the Leibniz integral rule several times to arrive at the following derivatives:

$$\frac{\partial q(k,b)}{\partial k} = \beta \alpha Ak^{\alpha-1} \int_{\epsilon^*}^{\infty} \frac{u'(C_1^i(\epsilon))}{u'(C_0^i)} e^\epsilon g(\epsilon) d\epsilon, \quad i = 1, 2$$

$$\frac{\partial q(k,b)}{\partial b} = -\beta \int_{\epsilon^*}^{\infty} \frac{u'(C_1^i(\epsilon))}{u'(C_0^i)} g(\epsilon) d\epsilon, \quad i = 1, 2$$

$$\frac{\partial p(k,b)}{\partial k} = \beta\alpha\frac{Ak^{\alpha-1}}{b}\int_{-\infty}^{\epsilon^*}\frac{u'(C_1^2(\epsilon))}{u'(C_0^2)}g(\epsilon)d\epsilon$$
$$\frac{\partial p(k,b)}{\partial b} = -\beta\frac{Ak^{\alpha}}{b^2}\int_{-\infty}^{\epsilon^*}\frac{u'(C_1^2(\epsilon))}{u'(C_0^2)}e^{\epsilon}g(\epsilon)d\epsilon$$

Special case: We confine ourselves to a special case in which both types of consumer hold positive equities so that $\frac{\partial q(k,b)}{\partial k}$ and $\frac{\partial q(k,b)}{\partial b}$ are related to rates of intertemporal substitution for both agents.

Substituting these partial derivatives into the above first-order conditions for k and b, respectively, we obtain the following versions of those first order conditions:

$$k: \quad -1 + \beta\alpha Ak^{\alpha-1}\int_{-\infty}^{\infty}\frac{u'(C_1^2(\epsilon))}{u'(C_0^2)}e^{\epsilon}g(\epsilon)d\epsilon = 0 \tag{2}$$

$$b: \quad \int_{\epsilon^*}^{\infty}\left(\frac{u'(C_1^1(\epsilon))}{u'(C_0^1)}\right)g(\epsilon)\,d\epsilon = \int_{\epsilon^*}^{\infty}\left(\frac{u'(C_1^2(\epsilon))}{u'(C_0^2)}\right)g(\epsilon)\,d\epsilon \tag{3}$$

where again recall that $\epsilon^*(k,b) \equiv \log\left(\frac{b}{Ak^{\alpha}}\right)$.

Taking $C_0^i, C_1^i(\epsilon)$ as given, these are two equations that we want to solve for the firm's optimal decisions k, b.

37.3 Equilibrium verification

On page 5 of BCG (2018), the authors say

If the price conjectures corresponding to the plan chosen by firms in equilibrium are correct, that is equal to the market prices $\breve{q}$ and $\breve{p}$, it is immediate to verify that the rationality of the conjecture coincides with the agents' Euler equations.

Here BCG are describing how they go about verifying that when they set little k, little b from the firm's first-order conditions equal to the big K, big B at the big C's that appear in the pricing functions, then

- consumers' Euler equations are satisfied if little c's are equated to Big C's
- firms' first-order necessary conditions for k, b are satisfied.
- $\breve{q} = q(K,B)$ and $\breve{p} = p(K,B)$.

37.4 Pseudo Code

Before displaying our Python code for computing a BCG incomplete markets equilibrium, we'll sketch some pseudo code that describes its logical flow.

Here goes:

1. Set upper and lower bounds for firm value as V_h and V_l, for capital as k_h and k_l, and for debt as b_h and b_l.
2. Conjecture firm value $V = \frac{1}{2}(V_h + V_l)$
3. Conjecture debt level $b = \frac{1}{2}(b_h + b_l)$.
4. Conjecture capital $k = \frac{1}{2}(k_h + k_l)$.
5. Compute the default threshold $\epsilon^* \equiv \log\left(\frac{b}{Ak^{\alpha}}\right)$.
6. (In this step we abuse notation by freezing V, k, b and in effect temporarily treating them as Big K, B values. Thus, in this step 6 little k, b are frozen at guessed at value of K, B.) Fixing the values of V, b and k, compute optimal choices of consumption c^i with consumers' FOCs. Assume that only agent 2 holds debt: $\xi^2 = b$ and that both agents hold equity: $0 < \theta^i < 1$ for $i = 1, 2$.

7. Set high and low bounds for equity holdings for agent 1 as θ^1_h and θ^1_l. Guess $\theta^1 = \frac{1}{2}(\theta^1_h + \theta^1_l)$, and $\theta^2 = 1 - \theta^1$. While $|\theta^1_h - \theta^1_l|$ is large:

   - Compute agent 1's valuation of the equity claim with a fixed-point iteration:

     $q_1 = \beta \int \frac{u'(c^1_1(\epsilon))}{u'(c^1_0)} d^e(k, b; \epsilon) g(\epsilon)\, d\epsilon$

     where

     $c^1_1(\epsilon) = w^1_1(\epsilon) + \theta^1 d^e(k, b; \epsilon)$

     and

     $c^1_0 = w^1_0 + \theta^1_0 V - q_1 \theta^1$

   - Compute agent 2's valuation of the bond claim with a fixed-point iteration:

     $p = \beta \int \frac{u'(c^2_1(\epsilon))}{u'(c^2_0)} d^b(k, b; \epsilon) g(\epsilon)\, d\epsilon$

     where

     $c^2_1(\epsilon) = w^2_1(\epsilon) + \theta^2 d^e(k, b; \epsilon) + b$

     and

     $c^2_0 = w^2_0 + \theta^2_0 V - q_1 \theta^2 - pb$

   - Compute agent 2's valuation of the equity claim with a fixed-point iteration:

     $q_2 = \beta \int \frac{u'(c^2_1(\epsilon))}{u'(c^2_0)} d^e(k, b; \epsilon) g(\epsilon)\, d\epsilon$

     where

     $c^2_1(\epsilon) = w^2_1(\epsilon) + \theta^2 d^e(k, b; \epsilon) + b$

     and

     $c^2_0 = w^2_0 + \theta^2_0 V - q_2 \theta^2 - pb$

   - If $q_1 > q_2$, Set $\theta_l = \theta^1$; otherwise, set $\theta_h = \theta^1$.
   - Repeat steps 6Aa through 6Ad until $|\theta^1_h - \theta^1_l|$ is small.

8. Set bond price as p and equity price as $q = \max(q_1, q_2)$.

9. Compute optimal choices of consumption:

$$\begin{aligned} c^1_0 &= w^1_0 + \theta^1_0 V - q\theta^1 \\ c^2_0 &= w^2_0 + \theta^2_0 V - q\theta^2 - pb \\ c^1_1(\epsilon) &= w^1_1(\epsilon) + \theta^1 d^e(k, b; \epsilon) \\ c^2_1(\epsilon) &= w^2_1(\epsilon) + \theta^2 d^e(k, b; \epsilon) + b \end{aligned}$$

10. (Here we confess to abusing notation again, but now in a different way. In step 7, we interpret frozen c^is as Big C^i. We do this to solve the firm's problem.) Fixing the values of c^i_0 and $c^i_1(\epsilon)$, compute optimal choices of capital k and debt level b using the firm's first order necessary conditions.

11. Compute deviations from the firm's FONC for capital k as:

    $kfoc = \beta\alpha A k^{\alpha-1} \left(\int \frac{u'(c^2_1(\epsilon))}{u'(c^2_0)} e^\epsilon g(\epsilon)\, d\epsilon \right) - 1$

   - If $kfoc > 0$, Set $k_l = k$; otherwise, set $k_h = k$.
   - Repeat steps 4 through 7A until $|k_h - k_l|$ is small.

12. Compute deviations from the firm's FONC for debt level b as:

    $bfoc = \beta \left[\int_{\epsilon^*}^{\infty} \left(\frac{u'(c^1_1(\epsilon))}{u'(c^1_0)} \right) g(\epsilon) \, d\epsilon - \int_{\epsilon^*}^{\infty} \left(\frac{u'(c^2_1(\epsilon))}{u'(c^2_0)} \right) g(\epsilon) \, d\epsilon \right]$

    - If $bfoc > 0$, Set $b_h = b$; otherwise, set $b_l = b$.
    - Repeat steps 3 through 7B until $|b_h - b_l|$ is small.

13. Given prices q and p from step 6, and the firm choices of k and b from step 7, compute the synthetic firm value:

    $V_x = -k + q + pb$

    - If $V_x > V$, then set $V_l = V$; otherwise, set $V_h = V$.
    - Repeat steps 1 through 8 until $|V_x - V|$ is small.

14. Ultimately, the algorithm returns equilibrium capital k^*, debt b^* and firm value V^*, as well as the following equilibrium values:

    - Equity holdings $\theta^{1,*} = \theta^1(k^*, b^*)$
    - Prices $q^* = q(k^*, b^*), \ p^* = p(k^*, b^*)$
    - Consumption plans $C^{1,*}_0 = c^1_0(k^*, b^*), \ C^{2,*}_0 = c^2_0(k^*, b^*), \ C^{1,*}_1(\epsilon) = c^1_1(k^*, b^*; \epsilon), \ C^{1,*}_1(\epsilon) = c^2_1(k^*, b^*; \epsilon)$.

37.5 Code

We create a Python class `BCG_incomplete_markets` to compute the equilibrium allocations of the incomplete market BCG model, given a set of parameter values.

The class includes the following methods, i.e., functions:

- `solve_eq`: solves the BCG model and returns the equilibrium values of capital k, debt b and firm value V, as well as
  - agent 1's equity holdings $\theta^{1,*}$
  - prices q^*, p^*
  - consumption plans $C^{1,*}_0, C^{2,*}_0, C^{1,*}_1(\epsilon), C^{2,*}_1(\epsilon)$.
- `eq_valuation`: inputs equilibrium consumpion plans C^* and outputs the following valuations for each pair of (k, b) in the grid:
  - the firm $V(k, b)$
  - the equity $q(k, b)$
  - the bond $p(k, b)$.

Parameters include:

- χ_1, χ_2: correlation parameter for agent 1 and 2. Default values are respectively 0 and 0.9.
- w^1_0, w^2_0: initial endowments. Default values are respectively 0.9 and 1.1.
- θ^1_0, θ^2_0: initial holding of the firm. Default values are 0.5.
- ψ: risk parameter. Default value is 3.
- α: Production function parameter. Default value is 0.6.
- A: Productivity of the firm. Default value is 2.5.
- μ, σ: Mean and standard deviation of the shock distribution. Default values are respectively -0.025 and 0.4

- β: Discount factor. Default value is 0.96.
- bound: Bound for truncated normal distribution. Default value is 3.

```
import pandas as pd
import numpy as np
from scipy.stats import norm
from scipy.stats import truncnorm
from scipy.integrate import quad
from scipy.optimize import bisect
from numba import njit
from interpolation import interp
```

```
class BCG_incomplete_markets:

    # init method or constructor
    def __init__(self,
                 ⍰1 = 0,
                 ⍰2 = 0.9,
                 w10 = 0.9,
                 w20 = 1.1,
                 ⍰10 = 0.5,
                 ⍰20 = 0.5,
                 ⍰1 = 3,
                 ⍰2 = 3,
                 ⍰ = 0.6,
                 A = 2.5,
                 ⍰ = -0.025,
                 ⍰ = 0.4,
                 ⍰ = 0.96,
                 bound = 3,
                 Vl = 0,
                 Vh = 0.5,
                 kbot = 0.01,
                 #ktop = (⍰*A)**(1/(1-⍰)),
                 ktop = 0.25,
                 bbot = 0.1,
                 btop = 0.8):

        #=========== Setup ===========#
        # Risk parameters
        self.⍰1 = ⍰1
        self.⍰2 = ⍰2

        # Other parameters
        self.⍰1 = ⍰1
        self.⍰2 = ⍰2
        self.⍰ = ⍰
        self.A = A
        self.⍰ = ⍰
        self.⍰ = ⍰
        self.⍰ = ⍰
        self.bound = bound

        # Bounds for firm value, capital, and debt
        self.Vl = Vl
        self.Vh = Vh
        self.kbot = kbot
```

(continues on next page)

(continued from previous page)

```
        #self.kbot = (⍰*A)**(1/(1-⍰))
        self.ktop = ktop
        self.bbot = bbot
        self.btop = btop

        # Utility
        self.u = njit(lambda c: (c**(1-⍰)) / (1-⍰))

        # Initial endowments
        self.w10 = w10
        self.w20 = w20
        self.w0 = w10 + w20

        # Initial holdings
        self.⍰10 = ⍰10
        self.⍰20 = ⍰20

        # Endowments at t=1
        self.w11 = njit(lambda ⍰: np.exp(-⍰1*⍰ - 0.5*(⍰1**2)*(⍰**2) + ⍰1*⍰))
        self.w21 = njit(lambda ⍰: np.exp(-⍰2*⍰ - 0.5*(⍰2**2)*(⍰**2) + ⍰2*⍰))
        self.w1 = njit(lambda ⍰: self.w11(⍰) + self.w21(⍰))

        # Truncated normal
        ta, tb = (-bound - ⍰) / ⍰, (bound - ⍰) / ⍰
        rv = truncnorm(ta, tb, loc=⍰, scale=⍰)
        ⍰_range = np.linspace(ta, tb, 1000000)
        pdf_range = rv.pdf(⍰_range)
        self.g = njit(lambda ⍰: interp(⍰_range, pdf_range, ⍰))


    #*************************************************************
    # Function: Solve for equilibrium of the BCG model
    #*************************************************************
    def solve_eq(self, print_crit=True):

        # Load parameters
        ⍰1 = self.⍰1
        ⍰2 = self.⍰2
        ⍰ = self.⍰
        A = self.A
        ⍰ = self.⍰
        bound = self.bound
        Vl = self.Vl
        Vh = self.Vh
        kbot = self.kbot
        ktop = self.ktop
        bbot = self.bbot
        btop = self.btop
        w10 = self.w10
        w20 = self.w20
        ⍰10 = self.⍰10
        ⍰20 = self.⍰20
        w11 = self.w11
        w21 = self.w21
        g = self.g

        # We need to find a fixed point on the value of the firm
```

(continues on next page)

(continued from previous page)

```
        V_crit = 1

        Y = njit(lambda □, fk: np.exp(□)*fk)
        intqq1 = njit(lambda □, fk, □1, □1, b: (w11(□) + □1*(Y(□, fk) - b))**(-
↪□1)*(Y(□, fk) - b)*g(□))
        intp1 = njit(lambda □, fk, □2, b: (Y(□, fk)/b)*(w21(□) + Y(□, fk))**(-
↪□2)*g(□))
        intp2 = njit(lambda □, fk, □2, □2, b: (w21(□) + □2*(Y(□, fk)-b) + b)**(-
↪□2)*g(□))
        intqq2 = njit(lambda □, fk, □2, □2, b: (w21(□) + □2*(Y(□, fk)-b) + b)**(-
↪□2)*(Y(□, fk) - b)*g(□))
        intk1 = njit(lambda □, fk, □2: (w21(□) + Y(□, fk))**(-□2)*np.exp(□)*g(□))
        intk2 = njit(lambda □, fk, □2, □2, b: (w21(□) + □2*(Y(□, fk)-b) + b)**(-
↪□2)*np.exp(□)*g(□))
        intB1 = njit(lambda □, fk, □1, □1, b: (w11(□) + □1*(Y(□, fk) - b))**(-
↪□1)*g(□))
        intB2 = njit(lambda □, fk, □2, □2, b: (w21(□) + □2*(Y(□, fk) - b) + b)**(-
↪□2)*g(□))

        while V_crit>1e-4:

            # We begin by adding the guess for the value of the firm to endowment
            V = (Vl+Vh)/2
            ww10 = w10 + □10*V
            ww20 = w20 + □20*V

            # Figure out the optimal level of debt
            bl = bbot
            bh = btop
            b_crit=1

            while b_crit>1e-5:

                # Setting the conjecture for debt
                b = (bl+bh)/2

                # Figure out the optimal level of capital
                kl = kbot
                kh = ktop
                k_crit=1

                while k_crit>1e-5:

                    # Setting the conjecture for capital
                    k = (kl+kh)/2

                    # Production
                    fk = A*(k**□)
#                     Y = lambda □: np.exp(□)*fk

                    # Compute integration threshold
                    epstar = np.log(b/fk)


                    #**************************************************************
                    # Compute the prices and allocations consistent with consumers'
                    # Euler equations
```

(continues on next page)

(continued from previous page)

```
                    #***************************************************************

                    # We impose the following:
                    # Agent 1 buys equity
                    # Agent 2 buys equity and all debt
                    # Agents trade such that prices converge

                    #========
                    # Agent 1
                    #========
                    # Holdings
                    □1 = 0
                    □1a = 0.3
                    □1b = 1

                    while abs(□1b - □1a) > 0.001:

                        □1 = (□1a + □1b) / 2

                        # qq1 is the equity price consistent with agent-1 Euler␣
↪Equation
                        ## Note: Price is in the date-0 budget constraint of the agent

                        ## First, compute the constant term that is not influenced by␣
↪q
                        ## that is, □E[u'(c^{1}_{1})d^{e}(k,B)]
#                         intqq1 = lambda □: (w11(□) + □1*(Y(□, fk) - b))**(-□1)*(Y(□,
↪ fk) - b)*g(□)
#                         const_qq1 = □ * quad(intqq1,epstar,bound)[0]

                        const_qq1 = □ * quad(intqq1,epstar,bound, args=(fk, □1, □1,␣
↪b))[0]


                        ## Second, iterate to get the equity price q
                        qq1l = 0
                        qq1h = ww10
                        diff = 1
                        while diff > 1e-7:
                            qq1 = (qq1l+qq1h)/2
                            rhs = const_qq1/((ww10-qq1*□1)**(-□1));
                            if (rhs > qq1):
                                qq1l = qq1
                            else:
                                qq1h = qq1
                            diff = abs(qq1l-qq1h)

                        #========
                        # Agent 2
                        #========
                        □2 = b - □1
                        □2 = 1 - □1

                        # p is the bond price consistent with agent-2 Euler Equation
                        ## Note: Price is in the date-0 budget constraint of the agent

                        ## First, compute the constant term that is not influenced by␣
↪p
```

(continues on next page)

(continued from previous page)

```
                        ## that is, □E[u'(c^{2}_{1})d^{b}(k,B)]
#                         intp1 = lambda □: (Y(□, fk)/b)*(w21(□) + Y(□, fk))**(-
↪□2)*g(□)
#                         intp2 = lambda □: (w21(□) + □2*(Y(□, fk)-b) + b)**(-□2)*g(□)
#                         const_p = □ * (quad(intp1,-bound,epstar)[0] + quad(intp2,
↪epstar,bound)[0])
                        const_p = □ * (quad(intp1,-bound,epstar, args=(fk, □2, b))[0]\
                                       + quad(intp2,epstar,bound, args=(fk, □2, □2,␣
↪b))[0])

                        ## iterate to get the bond price p
                        pl = 0
                        ph = ww20/b
                        diff = 1
                        while diff > 1e-7:
                            p = (pl+ph)/2
                            rhs = const_p/((ww20-qq1*□2-p*b)**(-□2))
                            if (rhs > p):
                                pl = p
                            else:
                                ph = p
                            diff = abs(pl-ph)

                        # qq2 is the equity price consistent with agent-2 Euler␣
↪Equation
#                         intqq2 = lambda □: (w21(□) + □2*(Y(□, fk)-b) + b)**(-
↪□2)*(Y(□, fk) - b)*g(□)
                        const_qq2 = □ * quad(intqq2,epstar,bound, args=(fk, □2, □2,␣
↪b))[0]
                        qq2l = 0
                        qq2h = ww20
                        diff = 1
                        while diff > 1e-7:
                            qq2 = (qq2l+qq2h)/2
                            rhs = const_qq2/((ww20-qq2*□2-p*b)**(-□2));
                            if (rhs > qq2):
                                qq2l = qq2
                            else:
                                qq2h = qq2
                            diff = abs(qq2l-qq2h)

                        # q be the maximum valuation for the equity among agents
                        ## This will be the equity price based on Makowski's criterion
                        q = max(qq1,qq2)

                        #=================
                        # Update holdings
                        #=================
                        if qq1 > qq2:
                            □1a = □1
                        else:
                            □1b = □1

                    #=================
                    # Get consumption
                    #=================
                    c10 = ww10 - q*□1
```

(continues on next page)

(continued from previous page)

```
                        c11 = lambda ⍰: w11(⍰) + ⍰1*max(Y(⍰, fk)-b,0)
                        c20 = ww20 - q*(1-⍰1) - p*b
                        c21 = lambda ⍰: w21(⍰) + (1-⍰1)*max(Y(⍰, fk)-b,0) + min(Y(⍰, fk),
↪b)


                        #**************************************************
                        # Compute the first order conditions for the firm
                        #**************************************************

                        #============
                        # Equity FOC
                        #============
                        # Only agent 2's IMRS is relevent
#                         intk1 = lambda ⍰: (w21(⍰) + Y(⍰, fk))**(-⍰2)*np.exp(⍰)*g(⍰)
#                         intk2 = lambda ⍰: (w21(⍰) + ⍰2*(Y(⍰, fk)-b) + b)**(-⍰2)*np.
↪exp(⍰)*g(⍰)
#                         kfoc_num = quad(intk1,-bound,epstar)[0] + quad(intk2,epstar,
↪bound)[0]
                        kfoc_num = quad(intk1,-bound,epstar, args=(fk, ⍰2))[0] +␣
↪quad(intk2,epstar,bound, args=(fk, ⍰2, ⍰2, b))[0]
                        kfoc_denom = (ww20- q*⍰2 - p*b)**(-⍰2)
                        kfoc = ⍰*⍰*A*(k**(⍰-1))*(kfoc_num/kfoc_denom) - 1

                        if (kfoc > 0):
                            kl = k
                        else:
                            kh = k
                        k_crit = abs(kh-kl)

                        if print_crit:
                            print("critical value of k: {:.5f}".format(k_crit))


                    #==========
                    # Bond FOC
                    #==========
#                     intB1 = lambda ⍰: (w11(⍰) + ⍰1*(Y(⍰, fk) - b))**(-⍰1)*g(⍰)
#                     intB2 = lambda ⍰: (w21(⍰) + ⍰2*(Y(⍰, fk) - b) + b)**(-⍰2)*g(⍰)

#                     bfoc1 = quad(intB1,epstar,bound)[0] / (ww10 - q*⍰1)**(-⍰1)
#                     bfoc2 = quad(intB2,epstar,bound)[0] / (ww20 - q*⍰2 - p*b)**(-⍰2)

                    bfoc1 = quad(intB1,epstar,bound, args=(fk, ⍰1, ⍰1, b))[0] / (ww10 -␣
↪q*⍰1)**(-⍰1)
                    bfoc2 = quad(intB2,epstar,bound, args=(fk, ⍰2, ⍰2, b))[0] / (ww20 -␣
↪q*⍰2 - p*b)**(-⍰2)
                    bfoc = bfoc1 - bfoc2

                    if (bfoc > 0):
                        bh = b
                    else:
                        bl = b
                    b_crit = abs(bh-bl)

                    if print_crit:
                        print("#=== critical value of b: {:.5f}".format(b_crit))
```

(continues on next page)

(continued from previous page)

```
        # Compute the value of the firm
        value_x = -k + q + p*b
        if (value_x > V):
             Vl = V
        else:
             Vh = V
        V_crit = abs(value_x-V)

        if print_crit:
            print("#====== critical value of V: {:.5f}".format(V_crit))

        print('k,b,p,q,kfoc,bfoc,epstar,V,V_crit')
        formattedList = ["%.3f" % member for member in [k,
                                                        b,
                                                        p,
                                                        q,
                                                        kfoc,
                                                        bfoc,
                                                        epstar,
                                                        V,
                                                        V_crit]]
        print(formattedList)

    #**********************************
    # Equilibrium values
    #**********************************

    # Return the results
    kss = k
    bss = b
    Vss = V
    qss = q
    pss = p
    c10ss = c10
    c11ss = c11
    c20ss = c20
    c21ss = c21
    □1ss = □1


    # Print the results
    print('finished')
   # print('k,b,p,q,kfoc,bfoc,epstar,V,V_crit')
    #formattedList = ["%.3f" % member for member in [kss,
    #                                                 bss,
    #                                                 pss,
    #                                                 qss,
    #                                                 kfoc,
    #                                                 bfoc,
    #                                                 epstar,
    #                                                 Vss,
    #                                                 V_crit]]
    #print(formattedList)

    return kss,bss,Vss,qss,pss,c10ss,c11ss,c20ss,c21ss,□1ss
```

(continues on next page)

(continued from previous page)

```
#*************************************************************
# Function: Equity and bond valuations by different agents
#*************************************************************
def valuations_by_agent(self,
                        c10, c11, c20, c21,
                        k, b):

    # Load parameters
    □1 = self.□1
    □2 = self.□2
    □ = self.□
    A = self.A
    □ = self.□
    bound = self.bound
    Vl = self.Vl
    Vh = self.Vh
    kbot = self.kbot
    ktop = self.ktop
    bbot = self.bbot
    btop = self.btop
    w10 = self.w10
    w20 = self.w20
    □10 = self.□10
    □20 = self.□20
    w11 = self.w11
    w21 = self.w21
    g = self.g

    # Get functions for IMRS/state price density
    IMRS1 = lambda □: □ * (c11(□)/c10)**(-□1)*g(□)
    IMRS2 = lambda □: □ * (c21(□)/c20)**(-□2)*g(□)

    # Production
    fk = A*(k**□)
    Y = lambda □: np.exp(□)*fk

    # Compute integration threshold
    epstar = np.log(b/fk)

    # Compute equity valuation with agent 1's IMRS
    intQ1 = lambda □: IMRS1(□)*(Y(□) - b)
    Q1 = quad(intQ1, epstar, bound)[0]

    # Compute bond valuation with agent 1's IMRS
    intP1 = lambda □: IMRS1(□)*Y(□)/b
    P1 = quad(intP1, -bound, epstar)[0] + quad(IMRS1, epstar, bound)[0]

    # Compute equity valuation with agent 2's IMRS
    intQ2 = lambda □: IMRS2(□)*(Y(□) - b)
    Q2 = quad(intQ2, epstar, bound)[0]

    # Compute bond valuation with agent 2's IMRS
    intP2 = lambda □: IMRS2(□)*Y(□)/b
    P2 = quad(intP2, -bound, epstar)[0] + quad(IMRS2, epstar, bound)[0]

    return Q1,Q2,P1,P2
```

(continues on next page)

(continued from previous page)

```
#**************************************************************
# Function: equilibrium valuations for firm, equity, bond
#**************************************************************
def eq_valuation(self, c10, c11, c20, c21, N=30):

    # Load parameters
    ⍰1 = self.⍰1
    ⍰2 = self.⍰2
    ⍰ = self.⍰
    A = self.A
    ⍰ = self.⍰
    bound = self.bound
    Vl = self.Vl
    Vh = self.Vh
    kbot = self.kbot
    ktop = self.ktop
    bbot = self.bbot
    btop = self.btop
    w10 = self.w10
    w20 = self.w20
    ⍰10 = self.⍰10
    ⍰20 = self.⍰20
    w11 = self.w11
    w21 = self.w21
    g = self.g

    # Create grids
    kgrid, bgrid = np.meshgrid(np.linspace(kbot,ktop,N),
                               np.linspace(bbot,btop,N))
    Vgrid = np.zeros_like(kgrid)
    Qgrid = np.zeros_like(kgrid)
    Pgrid = np.zeros_like(kgrid)

    # Loop: firm value
    for i in range(N):
        for j in range(N):

            # Get capital and debt
            k = kgrid[i,j]
            b = bgrid[i,j]

            # Valuations by each agent
            Q1,Q2,P1,P2 = self.valuations_by_agent(c10,
                                                   c11,
                                                   c20,
                                                   c21,
                                                   k,
                                                   b)

            # The prices will be the maximum of the valuations
            Q = max(Q1,Q2)
            P = max(P1,P2)

            # Compute firm value
            V = -k + Q + P*b
```

(continues on next page)

(continued from previous page)

```
                Vgrid[i,j] = V
                Qgrid[i,j] = Q
                Pgrid[i,j] = P

        return kgrid, bgrid, Vgrid, Qgrid, Pgrid
```

37.6 Examples

Below we show some examples computed with the class `BCG_incomplete markets`.

37.6.1 First example

In the first example, we set up an instance of the BCG incomplete markets model with default parameter values.

```
mdl = BCG_incomplete_markets()
kss,bss,Vss,qss,pss,c10ss,c11ss,c20ss,c21ss,□1ss = mdl.solve_eq(print_crit=False)
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.178', '0.503', '0.407', '0.092', '0.000', '-0.000', '-0.568', '0.250', '0.131']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.155', '0.487', '0.381', '0.073', '0.000', '0.000', '-0.518', '0.125', '0.021']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.144', '0.479', '0.368', '0.065', '0.000', '0.000', '-0.492', '0.062', '0.034']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.150', '0.484', '0.374', '0.069', '0.000', '-0.000', '-0.504', '0.094', '0.006']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.153', '0.486', '0.377', '0.071', '0.000', '-0.000', '-0.510', '0.109', '0.008']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '0.000', '0.000', '-0.508', '0.102', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.483', '0.375', '0.070', '-0.000', '0.000', '-0.507', '0.098', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.375', '0.070', '0.000', '-0.000', '-0.507', '0.100', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '0.000', '0.000', '-0.507', '0.101', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '0.000', '0.000', '-0.508', '0.101', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '-0.000', '-0.000', '-0.507', '0.101', '0.000']
finished
```

```
print(-kss+qss+pss*bss)
print(Vss)
print(□1ss)
```

```
0.10073912888808995
0.100830078125
0.98564453125
```

Python reports to us that the equilibrium firm value is $V = 0.101$, with capital $k = 0.151$ and debt $b = 0.484$.

Let's verify some things that have to be true if our algorithm has truly found an equilibrium.

Thus, let's see if the firm is actually maximizing its firm value given the equilibrium pricing function $q(k, b)$ for equity and $p(k, b)$ for bonds.

```
kgrid, bgrid, Vgrid, Qgrid, Pgrid = mdl.eq_valuation(c10ss, c11ss, c20ss, c21ss,N=30)

print('Maximum valuation of the firm value in the (k,B) grid: {:.5f}'.format(Vgrid.
↪max()))
print('Equilibrium firm value: {:.5f}'.format(Vss))
```

```
Maximum valuation of the firm value in the (k,B) grid: 0.10074
Equilibrium firm value: 0.10083
```

Up to the approximation involved in using a discrete grid, these numbers give us comfort that the firm does indeed seem to be maximizing its value at the top of the value hill on the (k, b) plane that it faces.

Below we will plot the firm's value as a function of k, b.

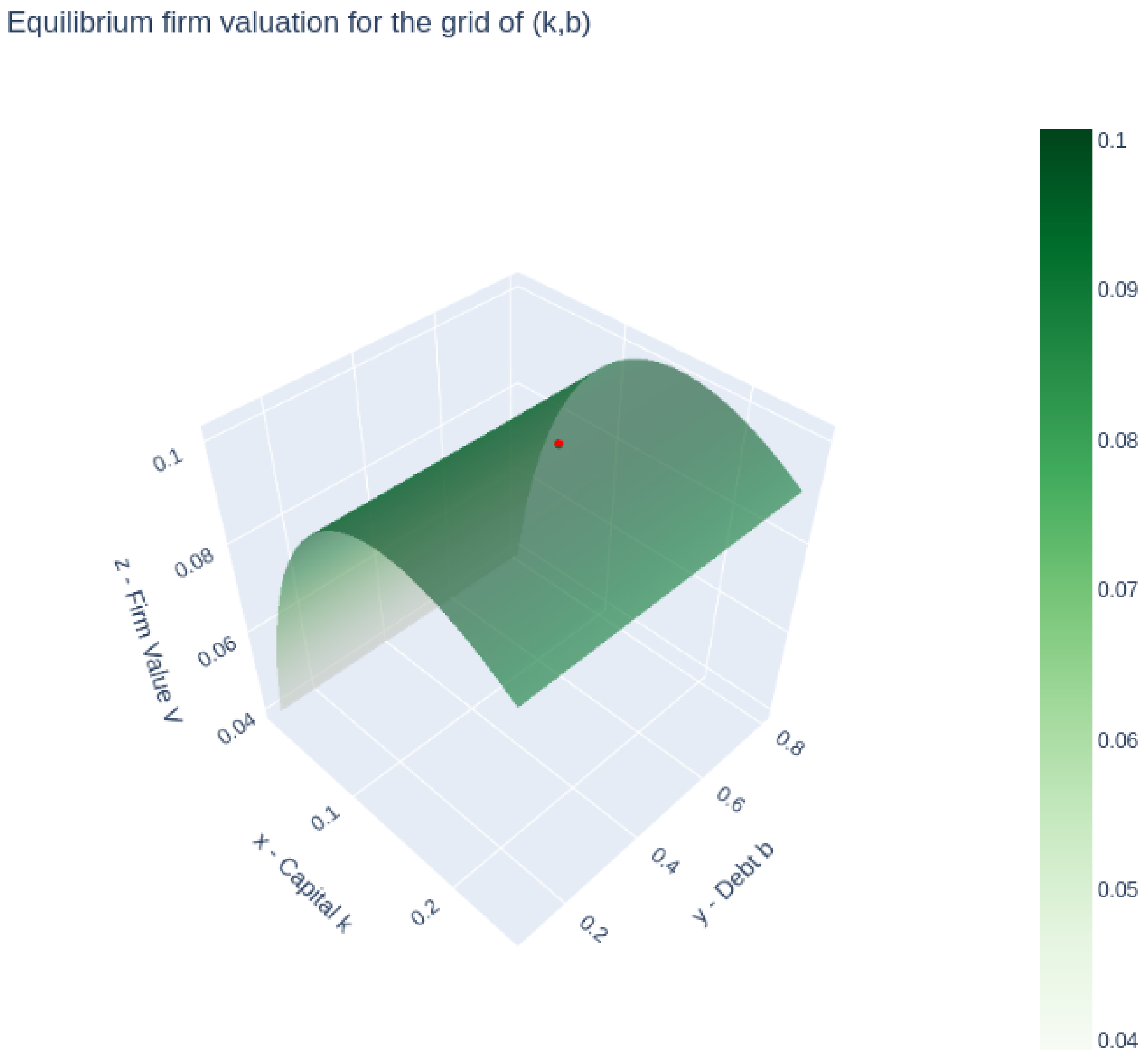
We'll also plot the equilibrium price functions $q(k, b)$ and $p(k, b)$.

```
from IPython.display import Image
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import plotly.graph_objs as go

# Firm Valuation
fig = go.Figure(data=[go.Scatter3d(x=[kss],
                                   y=[bss],
                                   z=[Vss],
                                   mode='markers',
                                   marker=dict(size=3, color='red')),
                      go.Surface(x=kgrid,
                                 y=bgrid,
                                 z=Vgrid,
                                 colorscale='Greens',opacity=0.6)])

fig.update_layout(scene = dict(
                    xaxis_title='x - Capital k',
                    yaxis_title='y - Debt b',
                    zaxis_title='z - Firm Value V',
                    aspectratio = dict(x=1,y=1,z=1)),
                  width=700,
```

(continues on next page)

(continued from previous page)

```
                   height=700,
                   margin=dict(l=50, r=50, b=65, t=90))
fig.update_layout(scene_camera=dict(eye=dict(x=1.5, y=-1.5, z=2)))
fig.update_layout(title='Equilibrium firm valuation for the grid of (k,b)')

# Export to PNG file
Image(fig.to_image(format="png"))
# fig.show() will provide interactive plot when running
# code locally
```



A Modigliani-Miller theorem?

The red dot in the above graph is **both** an equilibrium (b, k) chosen by a representative firm **and** the equilibrium B, K pair chosen by the aggregate of all firms.

Thus, **in equilibrium** it is true that

$$(b, k) = (B, K)$$

But an individual firm named $\zeta \in [0, 1]$ neither knows nor cares whether it sets $(b(\zeta), k(\zeta)) = (B, K)$.

Indeed the above graph has a ridge of $b(\zeta)$'s that also maximize the firm's value so long as it sets $k(\zeta) = K$.

Here it is important that the measure of firms that deviate from setting b at the red dot is very small – measure zero – so that B remains at the red dot even while one firm ζ deviates.

So within this equilibrium, there is a *qualified* Modigliani-Miller theorem that asserts that firm ζ's value is independent of how it mixes its financing between equity and bonds (so long as it is not what other firms do on average).

Thus, while an individual firm ζ's financial structure is indeterminate, the **market's** financial structure is determinant and sits at the red dot in the above graph.

This contrasts sharply with the *unqualified* Modigliani-Miller theorem descibed in the complete markets model in the lecture *Irrelevance of Capital Structure with Complete Markets*.

There the **market's** financial structure was indeterminate.

These subtle distinctions bear more thought and exploration.

So we will do some calculations to ferret out a sense in which the equilibrium $(k, b) = (K, B)$ outcome at the red dot in the above graph is **stable**.

In particular, we'll explore the consequences of some choices of $b = B$ that deviate from the red dot and ask whether firm ζ would want to remain at that b.

In more detail, here is what we'll do:

1. Obtain equilibrium values of capital and debt as $k^* = K$ and $b^* = B$, the red dot above.
2. Now fix k^* and let $b^{**} = b^* - e$ for some $e > 0$. Conjecture that big $K = k^*$ but big $B = b^{**}$.
3. Take K and B and compute intertermporal marginal rates of substitution (IMRS's) as we did before.
4. Taking the **new** IMRS to the firm's problem. Plot 3D surface for the valuations of the firm with this **new** IMRS.
5. Check if the value at k^*, b^{**} is at the top of this new 3D surface.
6. Repeat these calculations for $b^{**} = b^* + e$.

To conduct the above procedures, we create a function `off_eq_check` that inputs the BCG model instance parameters, equilibrium capital $K = k^*$ and debt $B = b^*$, and a perturbation of debt e.

The function outputs the fixed point firm values V^{**}, prices q^{**}, p^{**}, and consumption choices c^{**}.

Importantly, we relax the condition that only agent 2 holds bonds.

Now **both** agents can hold bonds, i.e., $0 \leq \xi^1 \leq B$ and $\xi^1 + \xi^2 = B$.

That implies the consumers' budget constraints are:

$$\begin{aligned} c^1_0 &= w^1_0 + \theta^1_0 V - q\theta^1 - p\xi^1 \\ c^2_0 &= w^2_0 + \theta^2_0 V - q\theta^2 - p\xi^2 \\ c^1_1(\epsilon) &= w^1_1(\epsilon) + \theta^1 d^e(k, b; \epsilon) + \xi^1 \\ c^2_1(\epsilon) &= w^2_1(\epsilon) + \theta^2 d^e(k, b; \epsilon) + \xi^2 \end{aligned}$$

The function also outputs agent 1's bond holdings ξ_1.

```
def off_eq_check(mdl,kss,bss,e=0.1):
    # Big K and big B
    k = kss
    b = bss + e

    # Load parameters
    □1 = mdl.□1
    □2 = mdl.□2
    □ = mdl.□
    A = mdl.A
    □ = mdl.□
    bound = mdl.bound
    Vl = mdl.Vl
    Vh = mdl.Vh
    kbot = mdl.kbot
    ktop = mdl.ktop
    bbot = mdl.bbot
    btop = mdl.btop
    w10 = mdl.w10
    w20 = mdl.w20
    □10 = mdl.□10
    □20 = mdl.□20
    w11 = mdl.w11
    w21 = mdl.w21
    g = mdl.g

    Y = njit(lambda □, fk: np.exp(□)*fk)
    intqq1 = njit(lambda □, fk, □1, □1, □1, b: (w11(□) + □1*(Y(□, fk) - b) + □1)**(-
↪□1)*(Y(□, fk) - b)*g(□))
    intpp1a = njit(lambda □, fk, □1, □1, b: (Y(□, fk)/b)*(w11(□) + Y(□, fk)/b*□1)**(-
↪□1)*g(□))
    intpp1b = njit(lambda □, fk, □1, □1, □1, b: (w11(□) + □1*(Y(□, fk)-b) + □1)**(-
↪□1)*g(□))
    intpp2a = njit(lambda □, fk, □2, □2, b: (Y(□, fk)/b)*(w21(□) + Y(□, fk)/b*□2)**(-
↪□2)*g(□))
    intpp2b = njit(lambda □, fk, □2, □2, □2, b: (w21(□) + □2*(Y(□, fk)-b) + □2)**(-
↪□2)*g(□))
    intqq2 = njit(lambda □, fk, □2, □2, b: (w21(□) + □2*(Y(□, fk)-b) + b)**(-□2)*(Y(□,
↪ fk) - b)*g(□))


    # Loop: Find fixed points V, q and p
    V_crit = 1
    while V_crit>1e-5:

        # We begin by adding the guess for the value of the firm to endowment
        V = (Vl+Vh)/2
        ww10 = w10 + □10*V
        ww20 = w20 + □20*V

        # Production
        fk = A*(k**□)
#           Y = lambda □: np.exp(□)*fk

        # Compute integration threshold
        epstar = np.log(b/fk)
```

(continues on next page)

(continued from previous page)

```
        #*************************************************************
        # Compute the prices and allocations consistent with consumers'
        # Euler equations
        #*************************************************************

        # We impose the following:
        # Agent 1 buys equity
        # Agent 2 buys equity and all debt
        # Agents trade such that prices converge

        #========
        # Agent 1
        #========
        # Holdings
        □1a = 0
        □1b = b/2
        p = 0.3

        while abs(□1b - □1a) > 0.001:

            □1 = (□1a + □1b) / 2
            □1a = 0.3
            □1b = 1

            while abs(□1b - □1a) > (0.001/b):

                □1 = (□1a + □1b) / 2

                # qq1 is the equity price consistent with agent-1 Euler Equation
                ## Note: Price is in the date-0 budget constraint of the agent

                ## First, compute the constant term that is not influenced by q
                ## that is, □E[u'(c^{1}_{1})d^{e}(k,B)]
#                 intqq1 = lambda □: (w11(□) + □1*(Y(□, fk) - b) + □1)**(-□1)*(Y(□,␣
↪fk) - b)*g(□)
#                 const_qq1 = □ * quad(intqq1,epstar,bound)[0]
                const_qq1 = □ * quad(intqq1,epstar,bound, args=(fk, □1, □1, □1, b))[0]

                ## Second, iterate to get the equity price q
                qq1l = 0
                qq1h = ww10
                diff = 1
                while diff > 1e-7:
                    qq1 = (qq1l+qq1h)/2
                    rhs = const_qq1/((ww10-qq1*□1-p*□1)**(-□1));
                    if (rhs > qq1):
                        qq1l = qq1
                    else:
                        qq1h = qq1
                    diff = abs(qq1l-qq1h)

                # pp1 is the bond price consistent with agent-2 Euler Equation
                ## Note: Price is in the date-0 budget constraint of the agent

                ## First, compute the constant term that is not influenced by p
                ## that is, □E[u'(c^{1}_{1})d^{b}(k,B)]
```

(continues on next page)

(continued from previous page)

```
#                  intpp1a = lambda □: (Y(□, fk)/b)*(w11(□) + Y(□, fk)/b*□1)**(-
↪□1)*g(□)
#                  intpp1b = lambda □: (w11(□) + □1*(Y(□, fk)-b) + □1)**(-□1)*g(□)
#                  const_pp1 = □ * (quad(intpp1a,-bound,epstar)[0] + quad(intpp1b,
↪epstar,bound)[0])
                 const_pp1 = □ * (quad(intpp1a,-bound,epstar, args=(fk, □1, □1, b))[0]␣
↪\
                                   + quad(intpp1b,epstar,bound, args=(fk, □1, □1, □1,␣
↪b))[0])

                 ## iterate to get the bond price p
                 pp1l = 0
                 pp1h = ww10/b
                 diff = 1
                 while diff > 1e-7:
                     pp1 = (pp1l+pp1h)/2
                     rhs = const_pp1/((ww10-qq1*□1-pp1*□1)**(-□1))
                     if (rhs > pp1):
                         pp1l = pp1
                     else:
                         pp1h = pp1
                     diff = abs(pp1l-pp1h)

                 #========
                 # Agent 2
                 #========
                 □2 = b - □1
                 □2 = 1 - □1

                 # pp2 is the bond price consistent with agent-2 Euler Equation
                 ## Note: Price is in the date-0 budget constraint of the agent

                 ## First, compute the constant term that is not influenced by p
                 ## that is, □E[u'(c^{2}_{1})d^{b}(k,B)]
#                  intpp2a = lambda □: (Y(□, fk)/b)*(w21(□) + Y(□, fk)/b*□2)**(-
↪□2)*g(□)
#                  intpp2b = lambda □: (w21(□) + □2*(Y(□, fk)-b) + □2)**(-□2)*g(□)
#                  const_pp2 = □ * (quad(intpp2a,-bound,epstar)[0] + quad(intpp2b,
↪epstar,bound)[0])
                 const_pp2 = □ * (quad(intpp2a,-bound,epstar, args=(fk, □2, □2, b))[0]␣
↪\
                                   + quad(intpp2b,epstar,bound, args=(fk, □2, □2, □2,␣
↪b))[0])

                 ## iterate to get the bond price p
                 pp2l = 0
                 pp2h = ww20/b
                 diff = 1
                 while diff > 1e-7:
                     pp2 = (pp2l+pp2h)/2
                     rhs = const_pp2/((ww20-qq1*□2-pp2*□2)**(-□2))
                     if (rhs > pp2):
                         pp2l = pp2
                     else:
                         pp2h = pp2
                     diff = abs(pp2l-pp2h)
```

(continues on next page)

(continued from previous page)

```
                # p be the maximum valuation for the bond among agents
                ## This will be the equity price based on Makowski's criterion
                p = max(pp1,pp2)


                # qq2 is the equity price consistent with agent-2 Euler Equation
#                 intqq2 = lambda □: (w21(□) + □2*(Y(□, fk)-b) + b)**(-□2)*(Y(□, fk) -
↪ b)*g(□)
#                 const_qq2 = □ * quad(intqq2,epstar,bound)[0]
                const_qq2 = □ * quad(intqq2,epstar,bound, args=(fk, □2, □2, b))[0]
                qq2l = 0
                qq2h = ww20
                diff = 1
                while diff > 1e-7:
                    qq2 = (qq2l+qq2h)/2
                    rhs = const_qq2/((ww20-qq2*□2-p*□2)**(-□2));
                    if (rhs > qq2):
                        qq2l = qq2
                    else:
                        qq2h = qq2
                    diff = abs(qq2l-qq2h)

                # q be the maximum valuation for the equity among agents
                ## This will be the equity price based on Makowski's criterion
                q = max(qq1,qq2)

                #=================
                # Update holdings
                #=================
                if qq1 > qq2:
                    □1a = □1
                else:
                    □1b = □1

                #print(p,q,□1,□1)

            if pp1 > pp2:
                □1a = □1
            else:
                □1b = □1


        #=================
        # Get consumption
        #=================
        c10 = ww10 - q*□1 - p*□1
        c11 = lambda □: w11(□) + □1*max(Y(□, fk)-b,0) + □1*min(Y(□, fk)/b,1)
        c20 = ww20 - q*(1-□1) - p*(b-□1)
        c21 = lambda □: w21(□) + (1-□1)*max(Y(□, fk)-b,0) + (b-□1)*min(Y(□, fk)/b,1)

        # Compute the value of the firm
        value_x = -k + q + p*b
        if (value_x > V):
            Vl = V
        else:
            Vh = V
        V_crit = abs(value_x-V)
```

(continues on next page)

(continued from previous page)

```
    return V,k,b,p,q,c10,c11,c20,c21,□1
```

Here is our strategy for checking *stability* of an equilibrium.

We use `off_eq_check` to obtain consumption plans for both agents at the conjectured big K and big B.

Then we input consumption plans into the function `eq_valuation` from the BCG model class and plot the agents' valuations associated with different choices of k and b.

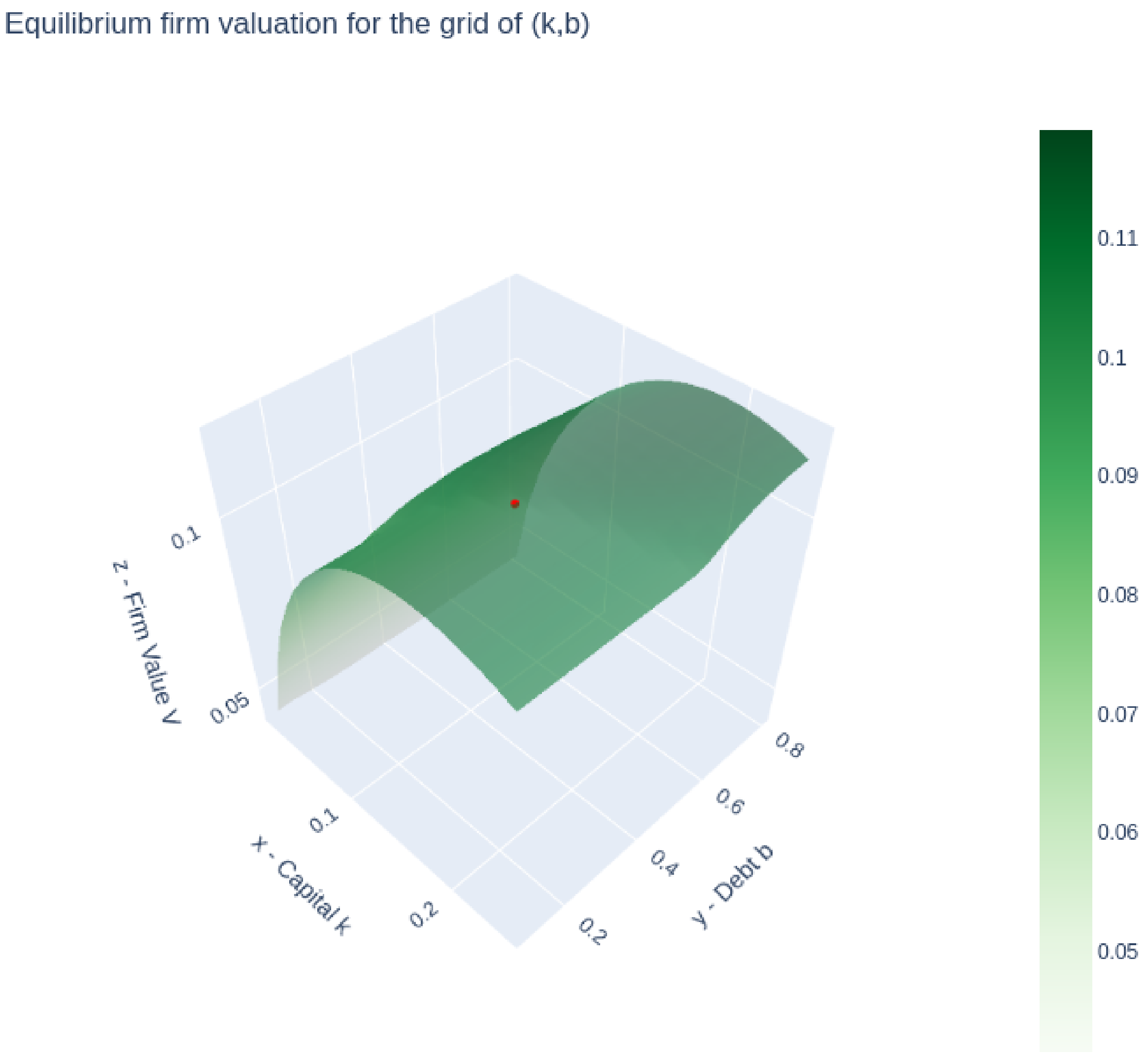
Our hunch is that (k^*, b^{**}) is **not** at the top of the firm valuation 3D surface so that the firm is **not** maximizing its value if it chooses $k = K = k^*$ and $b = B = b^{**}$.

That indicates that (k^*, b^{**}) is not an equilibrium capital structure for the firm.

We first check the case in which $b^{**} = b^* - e$ where $e = 0.1$:

```
#====================== Experiment 1 ======================#
Ve1,ke1,be1,pe1,qe1,c10e1,c11e1,c20e1,c21e1,□1e1 = off_eq_check(mdl,
                                                                   kss,
                                                                   bss,
                                                                   e=-0.1)

# Firm Valuation
kgride1, bgride1, Vgride1, Qgride1, Pgride1 = mdl.eq_valuation(c10e1, c11e1, c20e1,␣
↪c21e1,N=20)

print('Maximum valuation of the firm value in the (k,b) grid: {:.4f}'.format(Vgride1.
↪max()))
print('Equilibrium firm value: {:.4f}'.format(Ve1))

fig = go.Figure(data=[go.Scatter3d(x=[ke1],
                                   y=[be1],
                                   z=[Ve1],
                                   mode='markers',
                                   marker=dict(size=3, color='red')),
                      go.Surface(x=kgride1,
                                 y=bgride1,
                                 z=Vgride1,
                                 colorscale='Greens',opacity=0.6)])

fig.update_layout(scene = dict(
                    xaxis_title='x - Capital k',
                    yaxis_title='y - Debt b',
                    zaxis_title='z - Firm Value V',
                    aspectratio = dict(x=1,y=1,z=1)),
                  width=700,
                  height=700,
                  margin=dict(l=50, r=50, b=65, t=90))
fig.update_layout(scene_camera=dict(eye=dict(x=1.5, y=-1.5, z=2)))
fig.update_layout(title='Equilibrium firm valuation for the grid of (k,b)')


# Export to PNG file
Image(fig.to_image(format="png"))
# fig.show() will provide interactive plot when running
# code locally
```

```
Maximum valuation of the firm value in the (k,b) grid: 0.1191
Equilibrium firm value: 0.1118
```



In the above 3D surface of prospective firm valuations, the perturbed choice $(k^*, b^* - e)$, represented by the red dot, is not at the top.

The firm could issue more debts and attain a higher firm valuation from the market.

Therefore, $(k^*, b^* - e)$ would not be an equilibrium.

Next, we check for $b^{**} = b^* + e$.

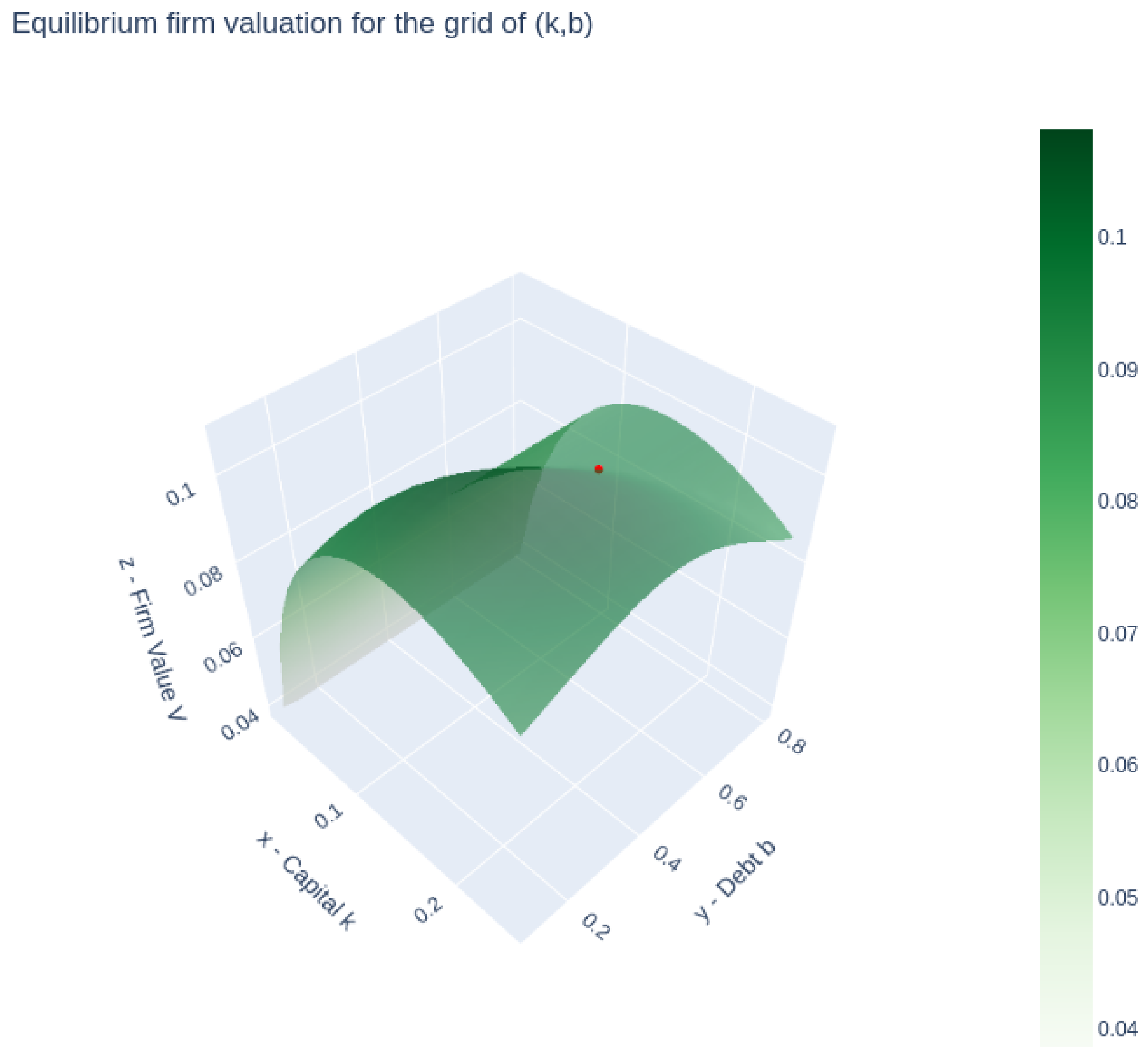
```
#====================== Experiment 2 ======================#
Ve2,ke2,be2,pe2,qe2,c10e2,c11e2,c20e2,c21e2,□1e2 = off_eq_check(mdl,
                                                                kss,
```

(continues on next page)

(continued from previous page)

```
                                                                    bss,
                                                                    e=0.1)

# Firm Valuation
kgride2, bgride2, Vgride2, Qgride2, Pgride2 = mdl.eq_valuation(c10e2, c11e2, c20e2,␣
↪c21e2,N=20)

print('Maximum valuation of the firm value in the (k,b) grid: {:.4f}'.format(Vgride2.
↪max()))
print('Equilibrium firm value: {:.4f}'.format(Ve2))

fig = go.Figure(data=[go.Scatter3d(x=[ke2],
                                   y=[be2],
                                   z=[Ve2],
                                   mode='markers',
                                   marker=dict(size=3, color='red')),
                      go.Surface(x=kgride2,
                                 y=bgride2,
                                 z=Vgride2,
                                 colorscale='Greens',opacity=0.6)])

fig.update_layout(scene = dict(
                    xaxis_title='x - Capital k',
                    yaxis_title='y - Debt b',
                    zaxis_title='z - Firm Value V',
                    aspectratio = dict(x=1,y=1,z=1)),
                  width=700,
                  height=700,
                  margin=dict(l=50, r=50, b=65, t=90))
fig.update_layout(scene_camera=dict(eye=dict(x=1.5, y=-1.5, z=2)))
fig.update_layout(title='Equilibrium firm valuation for the grid of (k,b)')

# Export to PNG file
Image(fig.to_image(format="png"))
# fig.show() will provide interactive plot when running
# code locally
```

```
Maximum valuation of the firm value in the (k,b) grid: 0.1082
Equilibrium firm value: 0.0974
```



In contrast to $(k^*, b^* - e)$, the 3D surface for $(k^*, b^* + e)$ now indicates that a firm would want o *decrease* its debt issuance to attain a higher valuation.

That incentive to deviate means that $(k^*, b^* + e)$ is not an equilibrium capital structure for the firm.

Interestingly, if consumers were to anticipate that firms would over-issue debt, i.e. $B > b^*$, then both types of consumer would want to hold corporate debt.

For example, $\xi^1 > 0$:

```
print('Bond holdings of agent 1: {:.3f}'.format(□1e2))
```

```
Bond holdings of agent 1: 0.039
```

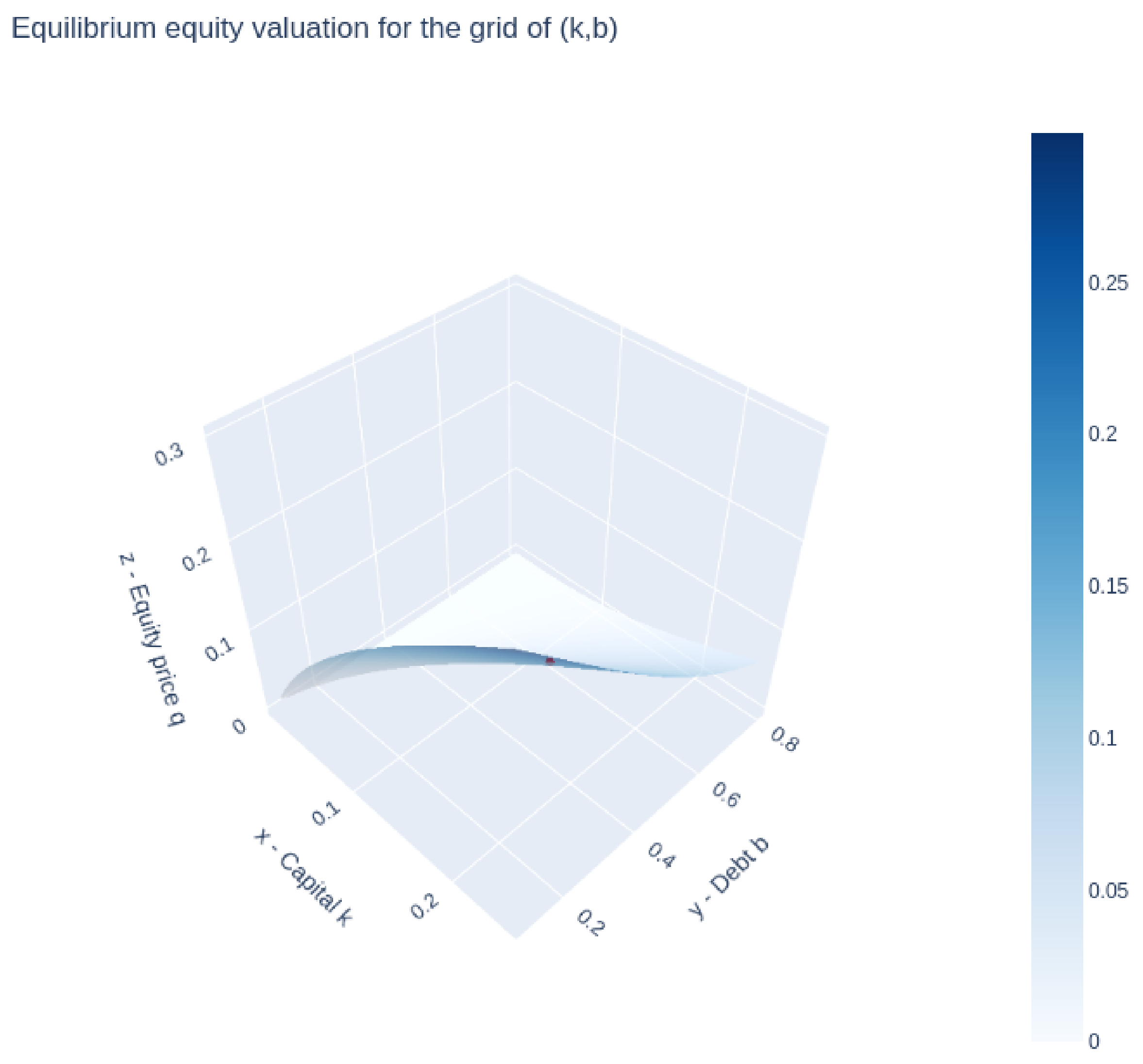
Our two *stability experiments* suggest that the equilibrium capital structure (k^*, b^*) is locally unique even though **at the equilibrium** an individual firm would be willing to deviate from the representative firms' equilibrium debt choice.

These experiments thus refine our discussion of the *qualified* Modigliani-Miller theorem that prevails in this example economy.

Equilibrium equity and bond price functions

It is also interesting to look at the equilibrium price functions $q(k, b)$ and $p(k, b)$ faced by firms in our rational expectations equilibrium.

```python
# Equity Valuation
fig = go.Figure(data=[go.Scatter3d(x=[kss],
                                   y=[bss],
                                   z=[qss],
                                   mode='markers',
                                   marker=dict(size=3, color='red')),
                      go.Surface(x=kgrid,
                                 y=bgrid,
                                 z=Qgrid,
                                 colorscale='Blues',opacity=0.6)])

fig.update_layout(scene = dict(
                    xaxis_title='x - Capital k',
                    yaxis_title='y - Debt b',
                    zaxis_title='z - Equity price q',
                    aspectratio = dict(x=1,y=1,z=1)),
                  width=700,
                  height=700,
                  margin=dict(l=50, r=50, b=65, t=90))
fig.update_layout(scene_camera=dict(eye=dict(x=1.5, y=-1.5, z=2)))
fig.update_layout(title='Equilibrium equity valuation for the grid of (k,b)')


# Export to PNG file
Image(fig.to_image(format="png"))
# fig.show() will provide interactive plot when running
# code locally
```

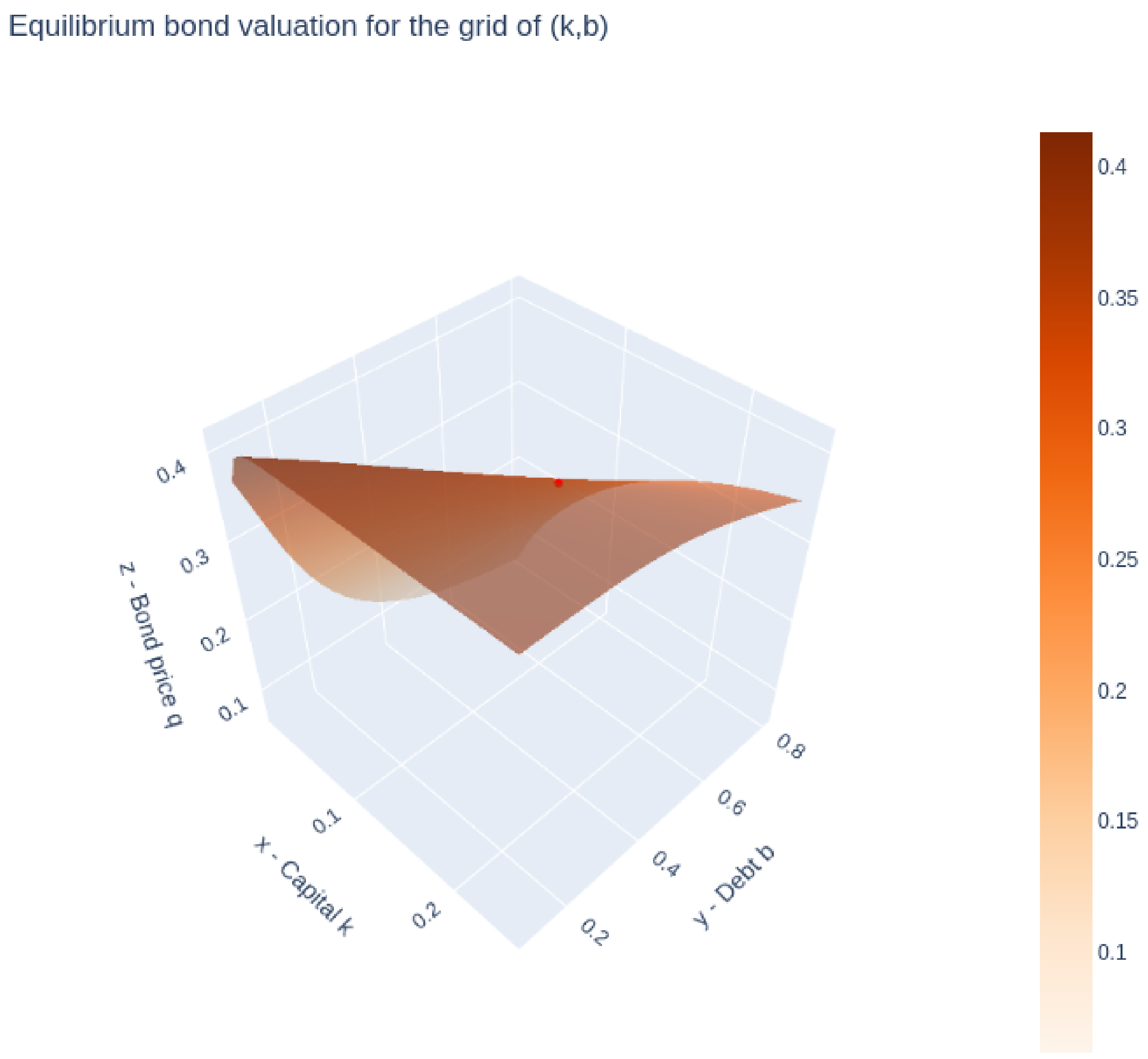


```
# Bond Valuation
fig = go.Figure(data=[go.Scatter3d(x=[kss],
                                   y=[bss],
                                   z=[pss],
                                   mode='markers',
                                   marker=dict(size=3, color='red')),
                      go.Surface(x=kgrid,
                                 y=bgrid,
                                 z=Pgrid,
                                 colorscale='Oranges',opacity=0.6)])

fig.update_layout(scene = dict(
                    xaxis_title='x - Capital k',
                    yaxis_title='y - Debt b',
```

(continues on next page)

(continued from previous page)

```
                    zaxis_title='z - Bond price q',
                    aspectratio = dict(x=1,y=1,z=1)),
                  width=700,
                  height=700,
                  margin=dict(l=50, r=50, b=65, t=90))
fig.update_layout(scene_camera=dict(eye=dict(x=1.5, y=-1.5, z=2)))
fig.update_layout(title='Equilibrium bond valuation for the grid of (k,b)')


# Export to PNG file
Image(fig.to_image(format="png"))
# fig.show() will provide interactive plot when running
# code locally
```



37.6.2 Comments on equilibrium pricing functions

The equilibrium pricing functions displayed above merit study and reflection.

They reveal the countervailing effects on a firm's valuations of bonds and equities that lie beneath the Modigliani-Miller ridge apparent in our earlier graph of an individual firm ζ's value as a function of $k(\zeta), b(\zeta)$.

37.6.3 Another example economy

We illustrate how the fraction of initial endowments held by agent 2, $w_0^2/(w_0^1+w_0^2)$ affects an equilibrium capital structure $(k, b) = (K, B)$ well as associated equilibrium allocations.

We are interested in how agents 1 and 2 value equity and bond.

$$Q^i = \beta \int \frac{u'(C_1^{i,*}(\epsilon))}{u'(C_0^{i,*})} d^e(k^*, b^*; \epsilon) g(\epsilon)\, d\epsilon$$

$$P^i = \beta \int \frac{u'(C_1^{i,*}(\epsilon))}{u'(C_0^{i,*})} d^b(k^*, b^*; \epsilon) g(\epsilon)\, d\epsilon$$

The function `valuations_by_agent` is used in calculating these valuations.

```python
# Lists for storage
wlist = []
klist = []
blist = []
qlist = []
plist = []
Vlist = []
tlist = []
q1list = []
q2list = []
p1list = []
p2list = []

# For loop: optimization for each endowment combination
for i in range(10):
    print(i)

    # Save fraction
    w10 = 0.9 - 0.05*i
    w20 = 1.1 + 0.05*i
    wlist.append(w20/(w10+w20))

    # Create the instance
    mdl = BCG_incomplete_markets(w10 = w10, w20 = w20, ktop = 0.5, btop = 2.5)

    # Solve for equilibrium
    kss,bss,Vss,qss,pss,c10ss,c11ss,c20ss,c21ss,⍰1ss = mdl.solve_eq(print_crit=False)

    # Store the equilibrium results
    klist.append(kss)
    blist.append(bss)
    qlist.append(qss)
    plist.append(pss)
    Vlist.append(Vss)
    tlist.append(⍰1ss)
```

(continues on next page)

(continued from previous page)

```
    # Evaluations of equity and bond by each agent
    Q1,Q2,P1,P2 = mdl.valuations_by_agent(c10ss, c11ss, c20ss, c21ss, kss, bss)

    # Save the valuations
    q1list.append(Q1)
    q2list.append(Q2)
    p1list.append(P1)
    p2list.append(P2)
```

```
0
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.178', '0.502', '0.407', '0.092', '-0.000', '-0.000', '-0.570', '0.250', '0.131']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.155', '0.487', '0.381', '0.073', '-0.001', '0.000', '-0.518', '0.125', '0.022']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.145', '0.480', '0.367', '0.065', '0.000', '-0.000', '-0.490', '0.062', '0.034']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.150', '0.484', '0.374', '0.069', '0.000', '0.000', '-0.504', '0.094', '0.006']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.153', '0.485', '0.378', '0.071', '0.000', '-0.000', '-0.511', '0.109', '0.008']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '-0.000', '-0.000', '-0.508', '0.102', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.483', '0.375', '0.070', '0.000', '-0.000', '-0.507', '0.098', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.375', '0.070', '-0.000', '0.000', '-0.506', '0.100', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '0.000', '0.000', '-0.507', '0.101', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '-0.000', '0.000', '-0.507', '0.101', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.151', '0.484', '0.376', '0.070', '0.000', '0.000', '-0.507', '0.101', '0.000']
finished
1
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.180', '0.544', '0.404', '0.081', '-0.000', '-0.000', '-0.498', '0.250', '0.130']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.158', '0.531', '0.378', '0.063', '-0.000', '-0.000', '-0.443', '0.125', '0.020']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.148', '0.525', '0.364', '0.055', '0.000', '-0.000', '-0.414', '0.062', '0.036']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.153', '0.528', '0.371', '0.059', '0.000', '0.000', '-0.428', '0.094', '0.008']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.156', '0.530', '0.374', '0.061', '-0.000', '-0.000', '-0.435', '0.109', '0.006']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.154', '0.529', '0.373', '0.060', '0.000', '-0.000', '-0.432', '0.102', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.155', '0.529', '0.373', '0.061', '0.000', '0.000', '-0.433', '0.105', '0.002']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.155', '0.529', '0.373', '0.060', '-0.000', '0.000', '-0.433', '0.104', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.154', '0.529', '0.373', '0.060', '-0.000', '0.000', '-0.433', '0.103', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.154', '0.529', '0.373', '0.060', '0.000', '-0.000', '-0.432', '0.103', '0.000']
finished
2
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.184', '0.590', '0.400', '0.070', '0.000', '0.000', '-0.427', '0.250', '0.128']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.162', '0.581', '0.373', '0.053', '0.000', '-0.000', '-0.366', '0.125', '0.017']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.152', '0.577', '0.359', '0.046', '-0.000', '-0.000', '-0.335', '0.062', '0.039']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.157', '0.579', '0.366', '0.049', '0.000', '0.000', '-0.351', '0.094', '0.011']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.159', '0.580', '0.369', '0.051', '-0.000', '-0.000', '-0.359', '0.109', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.158', '0.579', '0.368', '0.050', '-0.000', '-0.000', '-0.355', '0.102', '0.004']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.159', '0.579', '0.369', '0.051', '-0.000', '-0.000', '-0.357', '0.105', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.159', '0.580', '0.369', '0.051', '0.000', '0.000', '-0.358', '0.107', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.159', '0.580', '0.369', '0.051', '-0.000', '-0.000', '-0.358', '0.106', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.159', '0.580', '0.368', '0.051', '-0.000', '-0.000', '-0.357', '0.106', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.159', '0.580', '0.368', '0.051', '0.000', '-0.000', '-0.357', '0.106', '0.000']
finished
3
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.187', '0.642', '0.395', '0.059', '0.000', '0.000', '-0.354', '0.250', '0.125']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.166', '0.638', '0.366', '0.044', '-0.000', '-0.000', '-0.289', '0.125', '0.014']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.156', '0.637', '0.351', '0.037', '0.000', '-0.000', '-0.255', '0.062', '0.042']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.161', '0.637', '0.359', '0.040', '-0.000', '-0.000', '-0.273', '0.094', '0.014']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.164', '0.637', '0.363', '0.042', '-0.000', '-0.000', '-0.282', '0.109', '0.000']
finished
4
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.192', '0.702', '0.387', '0.049', '0.000', '-0.000', '-0.281', '0.250', '0.122']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.172', '0.704', '0.357', '0.035', '-0.000', '0.000', '-0.211', '0.125', '0.010']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.162', '0.706', '0.342', '0.029', '-0.000', '-0.000', '-0.173', '0.062', '0.046']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.167', '0.705', '0.350', '0.032', '-0.000', '-0.000', '-0.192', '0.094', '0.018']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.170', '0.705', '0.354', '0.033', '-0.000', '-0.000', '-0.202', '0.109', '0.004']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.171', '0.704', '0.356', '0.034', '0.000', '0.000', '-0.206', '0.117', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.170', '0.704', '0.355', '0.034', '-0.000', '-0.000', '-0.204', '0.113', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.170', '0.704', '0.355', '0.034', '-0.000', '0.000', '-0.205', '0.115', '0.002']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.170', '0.704', '0.355', '0.034', '-0.000', '0.000', '-0.205', '0.114', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.170', '0.704', '0.355', '0.034', '-0.000', '-0.000', '-0.205', '0.114', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.170', '0.704', '0.355', '0.034', '0.000', '0.000', '-0.204', '0.114', '0.000']
finished
5
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.198', '0.771', '0.377', '0.039', '-0.000', '-0.000', '-0.205', '0.250', '0.118']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.178', '0.781', '0.346', '0.027', '-0.000', '0.000', '-0.129', '0.125', '0.006']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.169', '0.788', '0.330', '0.022', '0.000', '-0.000', '-0.088', '0.062', '0.050']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.174', '0.784', '0.338', '0.024', '-0.000', '0.000', '-0.109', '0.094', '0.022']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.176', '0.783', '0.342', '0.026', '-0.000', '-0.000', '-0.119', '0.109', '0.008']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.177', '0.781', '0.344', '0.026', '0.000', '-0.000', '-0.125', '0.117', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.178', '0.781', '0.345', '0.027', '0.000', '0.000', '-0.127', '0.121', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.178', '0.782', '0.345', '0.026', '-0.000', '0.000', '-0.125', '0.119', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.177', '0.782', '0.345', '0.026', '-0.000', '-0.000', '-0.125', '0.118', '0.000']
finished
6
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.205', '0.851', '0.366', '0.030', '-0.000', '-0.000', '-0.126', '0.250', '0.114']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.186', '0.872', '0.333', '0.020', '0.000', '0.000', '-0.043', '0.125', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.177', '0.885', '0.316', '0.016', '0.000', '0.000', '0.001', '0.062', '0.055']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.181', '0.878', '0.324', '0.018', '-0.000', '0.000', '-0.022', '0.094', '0.027']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.184', '0.875', '0.328', '0.019', '0.000', '-0.000', '-0.033', '0.109', '0.013']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.185', '0.874', '0.330', '0.019', '0.000', '0.000', '-0.038', '0.117', '0.006']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.185', '0.873', '0.332', '0.019', '0.000', '0.000', '-0.041', '0.121', '0.002']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.186', '0.872', '0.332', '0.020', '-0.000', '-0.000', '-0.042', '0.123', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.186', '0.872', '0.332', '0.020', '-0.000', '-0.000', '-0.044', '0.124', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.186', '0.872', '0.332', '0.020', '-0.000', '0.000', '-0.043', '0.124', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.186', '0.872', '0.332', '0.020', '0.000', '0.000', '-0.043', '0.124', '0.000']
finished
7
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.212', '0.945', '0.351', '0.022', '0.000', '0.000', '-0.043', '0.250', '0.108']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.194', '0.980', '0.316', '0.014', '0.000', '0.000', '0.047', '0.125', '0.005']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.203', '0.961', '0.334', '0.018', '0.000', '0.000', '-0.000', '0.188', '0.052']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.199', '0.970', '0.325', '0.016', '-0.000', '0.000', '0.023', '0.156', '0.024']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.196', '0.975', '0.321', '0.015', '0.000', '0.000', '0.034', '0.141', '0.010']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.977', '0.319', '0.014', '0.000', '0.000', '0.041', '0.133', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.979', '0.318', '0.014', '-0.000', '0.000', '0.044', '0.129', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.978', '0.318', '0.014', '-0.000', '0.000', '0.042', '0.131', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.979', '0.318', '0.014', '-0.000', '0.000', '0.043', '0.130', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.978', '0.318', '0.014', '0.000', '-0.000', '0.042', '0.130', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.978', '0.318', '0.014', '0.000', '-0.000', '0.043', '0.130', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.195', '0.978', '0.318', '0.014', '-0.000', '0.000', '0.043', '0.130', '0.000']
finished
8
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.221', '1.056', '0.334', '0.016', '0.000', '0.000', '0.044', '0.250', '0.103']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.204', '1.110', '0.298', '0.009', '0.000', '0.000', '0.143', '0.125', '0.011']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.212', '1.081', '0.316', '0.012', '0.000', '0.000', '0.092', '0.188', '0.046']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.208', '1.095', '0.307', '0.010', '0.000', '0.000', '0.117', '0.156', '0.018']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.206', '1.102', '0.302', '0.010', '-0.000', '0.000', '0.130', '0.141', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.205', '1.107', '0.300', '0.009', '-0.000', '-0.000', '0.137', '0.133', '0.004']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.205', '1.105', '0.301', '0.009', '-0.000', '0.000', '0.133', '0.137', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.206', '1.104', '0.302', '0.010', '0.000', '0.000', '0.131', '0.139', '0.002']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.205', '1.104', '0.301', '0.009', '0.000', '-0.000', '0.132', '0.138', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.205', '1.104', '0.301', '0.009', '0.000', '-0.000', '0.133', '0.137', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.205', '1.104', '0.301', '0.009', '-0.000', '0.000', '0.133', '0.137', '0.000']
finished
9
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.231', '1.189', '0.315', '0.010', '0.000', '-0.000', '0.137', '0.250', '0.096']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.214', '1.269', '0.277', '0.005', '0.000', '-0.000', '0.247', '0.125', '0.018']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.222', '1.226', '0.296', '0.008', '-0.000', '0.000', '0.190', '0.188', '0.039']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.218', '1.247', '0.286', '0.006', '0.000', '-0.000', '0.218', '0.156', '0.011']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.216', '1.258', '0.282', '0.006', '-0.000', '-0.000', '0.233', '0.141', '0.003']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.217', '1.252', '0.284', '0.006', '-0.000', '-0.000', '0.225', '0.148', '0.004']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.216', '1.256', '0.283', '0.006', '-0.000', '0.000', '0.229', '0.145', '0.000']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.216', '1.256', '0.282', '0.006', '-0.000', '0.000', '0.231', '0.143', '0.002']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.216', '1.256', '0.282', '0.006', '0.000', '0.000', '0.230', '0.144', '0.001']
```

```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.216', '1.255', '0.283', '0.006', '0.000', '-0.000', '0.229', '0.144', '0.000']
```

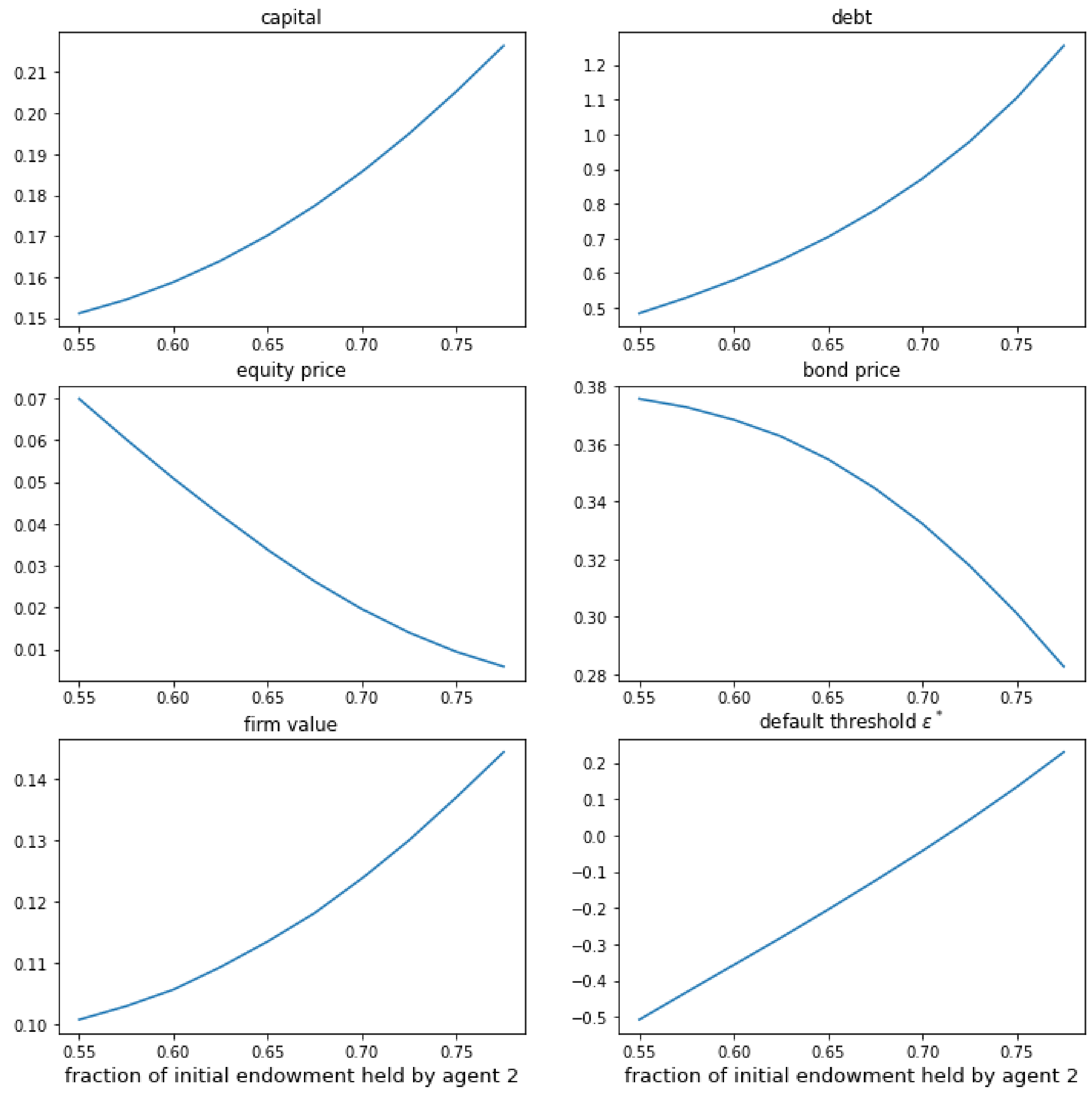
```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.216', '1.255', '0.283', '0.006', '-0.000', '0.000', '0.229', '0.144', '0.000']
finished
```

```python
# Plot
fig, ax = plt.subplots(3,2,figsize=(12,12))
ax[0,0].plot(wlist,klist)
ax[0,0].set_title('capital')
ax[0,1].plot(wlist,blist)
ax[0,1].set_title('debt')
ax[1,0].plot(wlist,qlist)
ax[1,0].set_title('equity price')
ax[1,1].plot(wlist,plist)
ax[1,1].set_title('bond price')
ax[2,0].plot(wlist,Vlist)
ax[2,0].set_title('firm value')
ax[2,0].set_xlabel('fraction of initial endowment held by agent 2',fontsize=13)

# Create a list of Default thresholds
A = mdl.A
⍰ = mdl.⍰
epslist = []
for i in range(len(wlist)):
    bb = blist[i]
    kk = klist[i]
    eps = np.log(bb/(A*kk**⍰))
```

(continues on next page)

(continued from previous page)

```
    epslist.append(eps)

# Plot (cont.)
ax[2,1].plot(wlist,epslist)
ax[2,1].set_title(r'default threshold $\epsilon^*$')
ax[2,1].set_xlabel('fraction of initial endowment held by agent 2',fontsize=13)
plt.show()
```



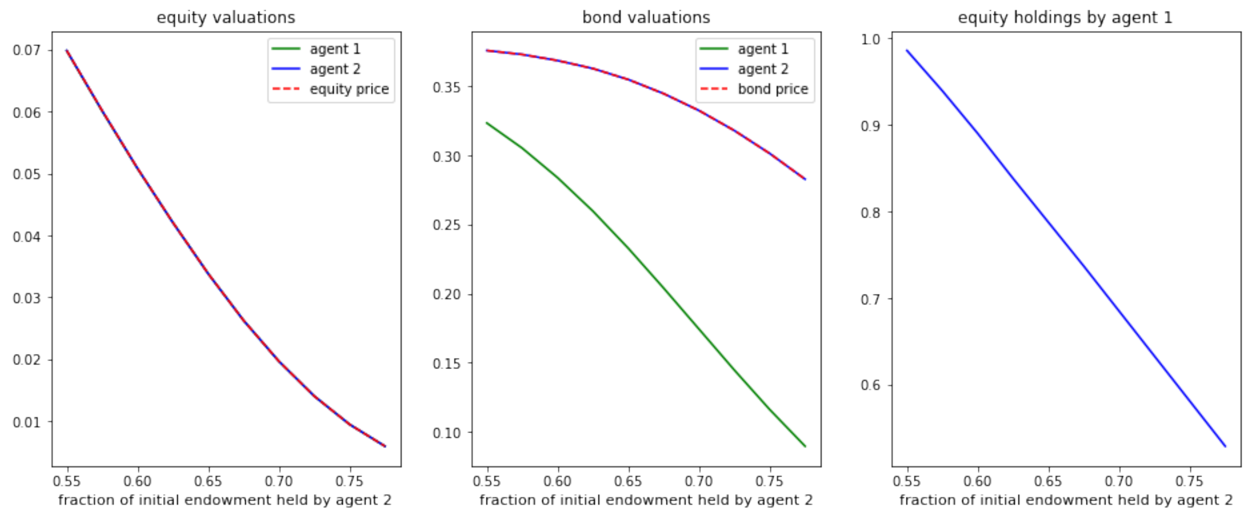
37.7 A picture worth a thousand words

Please stare at the above panels.

They describe how equilibrium prices and quantities respond to alterations in the structure of society's *hedging desires* across economies with different allocations of the initial endowment to our two types of agents.

Now let's see how the two types of agents value bonds and equities, keeping in mind that the type that values the asset highest determines the equilibrium price (and thus the pertinent set of Big C's).

```python
# Comparing the prices
fig, ax = plt.subplots(1,3,figsize=(16,6))

ax[0].plot(wlist,q1list,label='agent 1',color='green')
ax[0].plot(wlist,q2list,label='agent 2',color='blue')
ax[0].plot(wlist,qlist,label='equity price',color='red',linestyle='--')
ax[0].legend()
ax[0].set_title('equity valuations')
ax[0].set_xlabel('fraction of initial endowment held by agent 2',fontsize=11)

ax[1].plot(wlist,p1list,label='agent 1',color='green')
ax[1].plot(wlist,p2list,label='agent 2',color='blue')
ax[1].plot(wlist,plist,label='bond price',color='red',linestyle='--')
ax[1].legend()
ax[1].set_title('bond valuations')
ax[1].set_xlabel('fraction of initial endowment held by agent 2',fontsize=11)

ax[2].plot(wlist,tlist,color='blue')
ax[2].set_title('equity holdings by agent 1')
ax[2].set_xlabel('fraction of initial endowment held by agent 2',fontsize=11)

plt.show()
```



It is rewarding to stare at the above plots too.

In equilibrium, equity valuations are the same across the two types of agents but bond valuations are not.

Agents of type 2 value bonds more highly (they want more hedging).

Taken together with our earlier plot of equity holdings, these graphs confirm our earlier conjecture that while both type of agents hold equities, only agents of type 2 holds bonds.