Thomas J. Sargent & John Stachurski

38. Equilibrium Capital Structures with Incomplete Markets

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-orca
```

```
Requirement already satisfied: quantecon in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (0.5.2)
```

```
Requirement already satisfied: scipy>=1.0.0 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from quantecon)
(1.7.1)
Requirement already satisfied: sympy in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from quantecon) (1.9)
Requirement already satisfied: numba>=0.38 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from quantecon)
(0.54.1)
Requirement already satisfied: numpy in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from quantecon)
(1.20.3)
```

```
Requirement already satisfied: requests in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from quantecon)
(2.26.0)
Requirement already satisfied: llvmlite<0.38,>=0.37.0rc1 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages
(from numba>=0.38->quantecon) (0.37.0)
Requirement already satisfied: setuptools in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from numba>=0.38-
>quantecon) (58.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from
requests->quantecon) (2021.10.8)
Requirement already satisfied: idna<4,>=2.5 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from requests-
>quantecon) (3.2)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages
(from requests->quantecon) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from
requests->quantecon) (1.26.7)
Requirement already satisfied: mpmath>=0.19 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from sympy-
>quantecon) (1.2.1)
```

```
Requirement already satisfied: interpolation in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (2.2.1)
Requirement already satisfied: tempita>=0.5.2 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from
interpolation) (0.5.2)
Requirement already satisfied: numba>=0.47 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from
interpolation) (0.54.1)
Requirement already satisfied: scipy>=1.4.1 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from
interpolation) (1.7.1)
Requirement already satisfied: numpy>=1.18.1 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from
interpolation) (1.20.3)
Requirement already satisfied: llvmlite<0.38,>=0.37.0rc1 in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages
(from numba>=0.47->interpolation) (0.37.0)
Requirement already satisfied: setuptools in /usr/share/miniconda3/envs/quantecon/lib/python3.8/site-packages (from numba>=0.47-
>interpolation) (58.0.4)
```

```
Collecting package metadata (current_repodata.json): -
```

```
\
```

```
|
```

```
/
```

```
-
```

```
\
```

```
|
```

```
/
```

```
-
```



```
|
```

```
/
```

```
-
```

```
\
|
/
done
Solving environment: \
|
/
-
\
|
/
-
\
|
/
-
\
|
/
-
\
|
/
-
\
|
/
-
\
|
/
-
\
```

```
  |
  /
  -
  \
  |
  /
  -
  \
  |
  /
  -
  \
  |
  /
  -
  \
  |
  /
  -
  done
```

```
# All requested packages already installed.
```

38.1. Introduction

This is an extension of an earlier lecture **I**rrelevance of Capital Structure with Complete Markets (BCG_complete_mkts.html) 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 (zreferences.html#id29)] 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 **I**rrelevance of Capital Structure with Complete Markets (BCG_complete_mkts.html), 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 **I**rrelevance of Capital Structure with Complete Markets (BCG_complete_mkts.html)

- 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 (zreferences.html#id30)]
- 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 (zreferences.html#id30)], 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 (BCG_complete_mkts.html).

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 (BCG_complete_mkts.html).

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.)

38.1.1. Setup

We adopt specifications of preferences and technologies used by Bisin, Clemente, and Gottardi (2018) [BCG18 (zreferences.html#id29)] 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 $f(k)e^{\varepsilon}$ in period 1 from investing $k \geq 0$ in capital in period 0.
- period 1 endowments $w_1^i(\varepsilon)$ of the consumption good for agents $i = 1$ and $i = 2$.

38.1.2. Ownership

A consumer of type i is endowed with w_0^i units of the time 0 good and $w_1^i(\varepsilon)$ 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.

38.1.3. Measures of agents and firms

As in the companion lecture Irrelevance of Capital Structure with Complete Markets (BCG_complete_mkts.html) 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 $Ak^{\alpha}e^{\varepsilon}$ 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

$$C_0^i = \int_0^1 c_0^i(\omega) d\omega$$
$$C_1^i(\varepsilon) = \int_0^1 c_1^i(\varepsilon; \omega) d\omega$$

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.

38.1.4. Endowments

Per capital economy-wide endowments in periods 0 and 1 are

$$w_0 = w_0^1 + w_0^2$$
$$w_1(\varepsilon) = w_1^1(\varepsilon) + w_1^2(\varepsilon) \text{ in state } \varepsilon$$

38.1.5. Feasibility:

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

$$C_0^1 + C_0^2 = w_0^1 + w_0^2 - K$$
$$C_1^1(\varepsilon) + C_1^2(\varepsilon) = w_1^1(\varepsilon) + w_1^2(\varepsilon) + e^{\varepsilon}\int_0^1 f(k(\zeta))d\zeta, \quad k \geq 0$$

where $f(k) = Ak^{\alpha}, A > 0, \alpha \in (0,1)$.

38.1.6. Parameterizations

Following BCG, we shall employ the following parameterizations:

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

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

38.1.7. Preferences:

A consumer of type i orders period 0 consumption c_0^i and state ε-period 1 consumption $c^i(\varepsilon)$ by

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

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

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

38.1.8. 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.

38.2. 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 $e^{\varepsilon}Ak^{\alpha} < b$ at time 1, the firm defaults and its output is divided equally among bondholders.

Evidently, when the productivity shock $\varepsilon < \varepsilon^* = \log\left(\frac{b}{Ak^{\alpha}}\right)$, the firm defaults on its debt

Payoffs to equity and debt at date 1 as functions of the productivity shock ε are thus

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

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.

38.2.1. Consumers

Each consumer of type i is endowed with w_0^i of the time 0 consumption good, $w_1^i(\varepsilon)$ 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

$$d^e(K,B;\varepsilon) = \max\{e^{\varepsilon}AK^{\alpha} - B, 0\}$$
$$d^b(K,B;\varepsilon) = \min\left\{\frac{e^{\varepsilon}AK^{\alpha}}{B}, 1\right\}$$

Consumer i's optimization problem is

$$\begin{aligned}\max_{c_0^i,\theta^i,\xi^i,c_1^i(\varepsilon)} \quad & u(c_0^i)+\beta\int u(c^i(\varepsilon))g(\varepsilon)\,d\varepsilon \\ \text{subject to} \quad & c_0^i = w_0^i + \theta_0^i V - \check{q}\theta^i - \check{p}\xi^i, \\ & c_1^i(\varepsilon) = w_1^i(\varepsilon) + \theta^i d^e(K,B;\varepsilon) + \xi^i d^b(K,B;\varepsilon)\ \forall\,\varepsilon, \\ & \theta^i \geq 0, \xi^i \geq 0.\end{aligned}$$

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:

$$\begin{aligned}L^i :=& u(c_0^i)+\beta\int u(c^i(\varepsilon))g(\varepsilon)\,d\varepsilon \\ &+\lambda_0^i[w_0^i+\theta_0 V-\check{q}\theta^i-\check{p}\xi^i-c_0^i] \\ &+\beta\int\lambda_1^i(\varepsilon)\left[w_1^i(\varepsilon)+\theta^i d^e(K,B;\varepsilon)+\xi^i d^b(K,B;\varepsilon)-c_1^i(\varepsilon)\right]g(\varepsilon)\,d\varepsilon\end{aligned}$$

Consumer i's first-order necessary conditions for an optimum include:

$$\begin{aligned}c_0^i:&\quad u'(c_0^i)=\lambda_0^i \\ c_1^i(\varepsilon):&\quad u'(c_1^i(\varepsilon))=\lambda_1^i(\varepsilon) \\ \theta^i:&\quad \beta\int\lambda_1^i(\varepsilon)d^e(K,B;\varepsilon)g(\varepsilon)\,d\varepsilon\leq\lambda_0^i\check{q}\quad(=\text{ if }\theta^i>0) \\ \xi^i:&\quad \beta\int\lambda_1^i(\varepsilon)d^b(K,B;\varepsilon)g(\varepsilon)\,d\varepsilon\leq\lambda_0^i\check{p}\quad(=\text{ if }b^i>0)\end{aligned}$$

We can combine and rearrange consumer i's first-order conditions to become:

$$\begin{aligned}\check{q}&\geq\beta\int\frac{u'(c_1^i(\varepsilon))}{u'(c_0^i)}d^e(K,B;\varepsilon)g(\varepsilon)\,d\varepsilon\quad(=\text{ if }\theta^i>0) \\ \check{p}&\geq\beta\int\frac{u'(c_1^i(\varepsilon))}{u'(c_0^i)}d^b(K,B;\varepsilon)g(\varepsilon)\,d\varepsilon\quad(=\text{ if }b^i>0)\end{aligned}$$

These inequalities imply that in a symmetric rational expectations equilibrium consumption allocations and prices satisfy

$$\begin{aligned}\check{q}&=\max_i\beta\int\frac{u'(c_1^i(\varepsilon))}{u'(c_0^i)}d^e(K,B;\varepsilon)g(\varepsilon)\,d\varepsilon \\ \check{p}&=\max_i\beta\int\frac{u'(c_1^i(\varepsilon))}{u'(c_0^i)}d^b(K,B;\varepsilon)g(\varepsilon)\,d\varepsilon\end{aligned}$$

38.2.2. 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

$$\begin{aligned}q(k,b)&=\max_i\beta\int\frac{u'(C_1^i(\varepsilon))}{u'(C_0^i)}d^e(k,b;\varepsilon)g(\varepsilon)\,d\varepsilon \\ p(k,b)&=\max_i\beta\int\frac{u'(C_1^i(\varepsilon))}{u'(C_0^i)}d^b(k,b;\varepsilon)g(\varepsilon)\,d\varepsilon\end{aligned}$$

where the payoff functions are described by equations (38.1).

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.

38.2.3. 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

$$q(k,b) = \beta \int \frac{u'(C_1^1(\varepsilon))}{u'(C_0^1)} d^e(k,b;\varepsilon) g(\varepsilon)\, d\varepsilon = \beta \int \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} d^e(k,b;\varepsilon) g(\varepsilon)\, d\varepsilon$$
$$p(k,b) = \beta \int \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} d^b(k,b;\varepsilon) g(\varepsilon)\, d\varepsilon$$

Recall that $\varepsilon^*(k,b) \equiv \log\left(\frac{b}{Ak^\alpha}\right)$ is a firm's default threshold.

We can rewrite the pricing functions as:

$$q(k,b) = \beta \int_{\varepsilon^*}^{\infty} \frac{u'(C_1^i(\varepsilon))}{u'(C_0^i)} (e^\varepsilon A k^\alpha - b) g(\varepsilon)\, d\varepsilon, \quad i = 1,2$$
$$p(k,b) = \beta \int_{-\infty}^{\varepsilon^*} \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} \left(\frac{e^\varepsilon A k^\alpha}{b}\right) g(\varepsilon)\, d\varepsilon + \beta \int_{\varepsilon^*}^{\infty} \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} g(\varepsilon)\, d\varepsilon$$

38.2.3.1. Firm's optimization problem

The firm's optimization problem is

$$V \equiv \max_{k,b} \{-k + q(k,b) + p(k,b)b\}$$

The firm's first-order necessary conditions with respect to k and b, respectively, are

$$k: \quad -1 + \frac{\partial q(k,b)}{\partial k} + b\frac{\partial p(q,b)}{\partial k} = 0$$
$$b: \quad \frac{\partial q(k,b)}{\partial b} + p(k,b) + b\frac{\partial p(k,b)}{\partial b} = 0$$

We use the Leibniz integral rule several times to arrive at the following derivatives:

$$\frac{\partial q(k,b)}{\partial k} = \beta\alpha A k^{\alpha-1} \int_{\varepsilon^*}^{\infty} \frac{u'(C_1^i(\varepsilon))}{u'(C_0^i)} e^\varepsilon g(\varepsilon) d\varepsilon, \quad i = 1,2$$

$$\frac{\partial q(k,b)}{\partial b} = -\beta \int_{\varepsilon^*}^{\infty} \frac{u'(C_1^i(\varepsilon))}{u'(C_0^i)} g(\varepsilon) d\varepsilon, \quad i = 1,2$$

$$\frac{\partial p(k,b)}{\partial k} = \beta\alpha \frac{A k^{\alpha-1}}{b} \int_{-\infty}^{\varepsilon^*} \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} g(\varepsilon) d\varepsilon$$

$$\frac{\partial p(k,b)}{\partial b} = -\beta \frac{A k^\alpha}{b^2} \int_{-\infty}^{\varepsilon^*} \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} e^\varepsilon g(\varepsilon) d\varepsilon$$

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:

$$k: \quad -1 + \beta\alpha A k^{\alpha-1} \int_{-\infty}^{\infty} \frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)} e^\varepsilon g(\varepsilon) d\varepsilon = 0$$

$$b: \quad \int_{\varepsilon^*}^{\infty} \left(\frac{u'(C_1^1(\varepsilon))}{u'(C_0^1)}\right) g(\varepsilon)\, d\varepsilon = \int_{\varepsilon^*}^{\infty} \left(\frac{u'(C_1^2(\varepsilon))}{u'(C_0^2)}\right) g(\varepsilon)\, d\varepsilon$$

where again recall that $\varepsilon^*(k,b) \equiv \log\left(\frac{b}{Ak^\alpha}\right)$.

Taking $C_0^i, C_1^i(\varepsilon)$ as given, these are two equations that we want to solve for the firm's optimal decisions k, b.

38.3. 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 $\check{q}$ and $\check{p}$, 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.
- $\check{q} = q(K,B)$ and $\check{p} = p(K,B)$.

38.4. 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 $\varepsilon^* \equiv \log\left(\frac{b}{Ak^\alpha}\right)$.
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_h^1 and θ_l^1. Guess $\theta^1 = \frac{1}{2}(\theta_h^1 + \theta_l^1)$, and $\theta^2 = 1 - \theta^1$. While $|\theta_h^1 - \theta_l^1|$ is large:
    - Compute agent 1's valuation of the equity claim with a fixed-point iteration:

      $$q_1 = \beta \int \frac{u'(c_1^1(\varepsilon))}{u'(c_0^1)} d^e(k, b; \varepsilon) g(\varepsilon)\, d\varepsilon$$

      where

      $$c_1^1(\varepsilon) = w_1^1(\varepsilon) + \theta^1 d^e(k, b; \varepsilon)$$

      and

      $$c_0^1 = w_0^1 + \theta_0^1 V - q_1 \theta^1$$

    - Compute agent 2's valuation of the bond claim with a fixed-point iteration:

      $$p = \beta \int \frac{u'(c_1^2(\varepsilon))}{u'(c_0^2)} d^b(k, b; \varepsilon) g(\varepsilon)\, d\varepsilon$$

      where

      $$c_1^2(\varepsilon) = w_1^2(\varepsilon) + \theta^2 d^e(k, b; \varepsilon) + b$$

      and

      $$c_0^2 = w_0^2 + \theta_0^2 V - q_1 \theta^2 - pb$$

    - Compute agent 2's valuation of the equity claim with a fixed-point iteration:

      $$q_2 = \beta \int \frac{u'(c_1^2(\varepsilon))}{u'(c_0^2)} d^e(k, b; \varepsilon) g(\varepsilon)\, d\varepsilon$$

      where

      $$c_1^2(\varepsilon) = w_1^2(\varepsilon) + \theta^2 d^e(k, b; \varepsilon) + b$$

      and

      $$c_0^2 = w_0^2 + \theta_0^2 V - q_2 \theta^2 - pb$$

    - If $q_1 > q_2$, Set $\theta_l = \theta^1$; otherwise, set $\theta_h = \theta^1$.
    - Repeat steps 6Aa through 6Ad until $|\theta_h^1 - \theta_l^1|$ is small.
8. Set bond price as p and equity price as $q = \max(q_1, q_2)$.
9. Compute optimal choices of consumption:

$$\begin{aligned} c_0^1 &= w_0^1 + \theta_0^1 V - q\theta^1 \\ c_0^2 &= w_0^2 + \theta_0^2 V - q\theta^2 - pb \\ c_1^1(\varepsilon) &= w_1^1(\varepsilon) + \theta^1 d^e(k, b; \varepsilon) \\ c_1^2(\varepsilon) &= w_1^2(\varepsilon) + \theta^2 d^e(k, b; \varepsilon) + b \end{aligned}$$

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_0^i and $c_1^i(\varepsilon)$, 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:

$$kfoc = \beta \alpha A k^{\alpha - 1} \left(\int \frac{u'(c_1^2(\varepsilon))}{u'(c_0^2)} e^e g(\varepsilon)\, d\varepsilon \right) - 1$$

    - 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:

$$bfoc = \beta \left[\int_{\varepsilon^*}^{\infty} \left(\frac{u'(c_1^1(\varepsilon))}{u'(c_0^1)} \right) g(\varepsilon)\, d\varepsilon - \int_{\varepsilon^*}^{\infty} \left(\frac{u'(c_1^2(\varepsilon))}{u'(c_0^2)} \right) g(\varepsilon)\, d\varepsilon \right]$$

    - 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:

$$V_s = -k + q + pb$$

    - If $V_s > V$, then set $V_l = V$; otherwise, set $V_h = V$.
    - Repeat steps 1 through 8 until $|V_s - V|$ 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
      $C_0^{1,*} = c_0^1(k^*, b^*),\ C_0^{2,*} = c_0^2(k^*, b^*),\ C_1^{1,*}(\varepsilon) = c_1^1(k^*, b^*; \varepsilon),\ C_1^{1,*}(\varepsilon) = c_1^2(k^*, b^*; \varepsilon)$.

38.5. 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 $C_0^{1,*}, C_0^{2,*}, C_1^{1,*}(\varepsilon), C_1^{2,*}(\varepsilon)$.

- `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_0^1, w_0^2: initial endowments. Default values are respectively 0.9 and 1.1.
- θ_0^1, θ_0^2: 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,
                 χ1 = 0,
                 χ2 = 0.9,
                 w10 = 0.9,
                 w20 = 1.1,
                 θ10 = 0.5,
                 θ20 = 0.5,
                 ψ1 = 3,
                 ψ2 = 3,
                 α = 0.6,
                 A = 2.5,
                 μ = -0.025,
                 σ = 0.4,
                 β = 0.96,
                 bound = 3,
                 Vl = 0,
                 Vh = 0.5,
                 kbot = 0.01,
                 #ktop = (α*A)**(1/(1-α)),
                 ktop = 0.25,
                 bbot = 0.1,
                 btop = 0.8):

        #=========== Setup ===========#
        # Risk parameters
        self.χ1 = χ1
        self.χ2 = χ2

        # Other parameters
        self.ψ1 = ψ1
        self.ψ2 = ψ2
        self.α = α
        self.A = A
        self.μ = μ
        self.σ = σ
        self.β = β
        self.bound = bound

        # Bounds for firm value, capital, and debt
        self.Vl = Vl
        self.Vh = Vh
        self.kbot = kbot
        #self.kbot = (α*A)**(1/(1-α))
        self.ktop = ktop
        self.bbot = bbot
        self.btop = btop

        # Utility
        self.u = njit(lambda c: (c**(1-ψ)) / (1-ψ))

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

        # Initial holdings
        self.θ10 = θ10
        self.θ20 = θ20

        # Endowments at t=1
        self.w11 = njit(lambda ϵ: np.exp(-χ1*μ - 0.5*(χ1**2)*(σ**2) + χ1*ϵ))
        self.w21 = njit(lambda ϵ: np.exp(-χ2*μ - 0.5*(χ2**2)*(σ**2) + χ2*ϵ))
        self.w1 = njit(lambda ϵ: self.w11(ϵ) + self.w21(ϵ))

        # Truncated normal
        ta, tb = (-bound - μ) / σ, (bound - μ) / σ
        rv = truncnorm(ta, tb, loc=μ, scale=σ)
        ϵ_range = np.linspace(ta, tb, 1000000)
        pdf_range = rv.pdf(ϵ_range)
        self.g = njit(lambda ϵ: interp(ϵ_range, pdf_range, ϵ))


    #***********************************************************
    # Function: Solve for equilibrium of the BCG model
    #***********************************************************
    def solve_eq(self, print_crit=True):

        # Load parameters
        ψ1 = self.ψ1
        ψ2 = self.ψ2
        α = self.α
        A = self.A
        β = self.β
        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
        θ10 = self.θ10
        θ20 = self.θ20
        w11 = self.w11
        w21 = self.w21
        g = self.g

        # We need to find a fixed point on the value of the firm
        V_crit = 1

        Y = njit(lambda ϵ, fk: np.exp(ϵ)*fk)
        intqq1 = njit(lambda ϵ, fk, θ1, ψ1, b: (w11(ϵ) + θ1*(Y(ϵ, fk) - b))**(-ψ1)*(Y(ϵ, fk) - b)*g(ϵ))
        intp1 = njit(lambda ϵ, fk, ψ2, b: (Y(ϵ, fk)/b)*(w21(ϵ) + Y(ϵ, fk))**(-ψ2)*g(ϵ))
        intp2 = njit(lambda ϵ, fk, θ2, ψ2, b: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*g(ϵ))
        intqq2 = njit(lambda ϵ, fk, θ2, ψ2, b: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*(Y(ϵ, fk) - b)*g(ϵ))
        intk1 = njit(lambda ϵ, fk, ψ2: (w21(ϵ) + Y(ϵ, fk))**(-ψ2)*np.exp(ϵ)*g(ϵ))
        intk2 = njit(lambda ϵ, fk, θ2, ψ2, b: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*np.exp(ϵ)*g(ϵ))
        intB1 = njit(lambda ϵ, fk, θ1, ψ1, b: (w11(ϵ) + θ1*(Y(ϵ, fk) - b))**(-ψ1)*g(ϵ))
        intB2 = njit(lambda ϵ, fk, θ2, ψ2, b: (w21(ϵ) + θ2*(Y(ϵ, fk) - b) + b)**(-ψ2)*g(ϵ))

        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 + θ10*V
            ww20 = w20 + θ20*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**α)
#                     Y = lambda ϵ: np.exp(ϵ)*fk

                    # Compute integration threshold
                    epstar = np.log(b/fk)


                    #************************************************************
                    # 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
                    ξ1 = 0
                    θ1a = 0.3
                    θ1b = 1

                    while abs(θ1b - θ1a) > 0.001:

                        θ1 = (θ1a + θ1b) / 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, βE[u'(c^{1}_{1})d^{e}(k,B)]
#                         intqq1 = lambda ϵ: (w11(ϵ) + θ1*(Y(ϵ, fk) - b))**(-ψ1)*(Y(ϵ, fk) - b)*g(ϵ)
#                         const_qq1 = β * quad(intqq1,epstar,bound)[0]

                        const_qq1 = β * quad(intqq1,epstar,bound, args=(fk, θ1, ψ1, 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*θ1)**(-ψ1));
                            if (rhs > qq1):
                                qq1l = qq1
                            else:
                                qq1h = qq1
                            diff = abs(qq1l-qq1h)

                        #========
                        # Agent 2
                        #========
                        ξ2 = b - ξ1
                        θ2 = 1 - θ1

                        # 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
                        ## that is, βE[u'(c^{2}_{1})d^{b}(k,B)]
#                         intp1 = lambda ϵ: (Y(ϵ, fk)/b)*(w21(ϵ) + Y(ϵ, fk))**(-ψ2)*g(ϵ)
#                         intp2 = lambda ϵ: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*g(ϵ)
#                         const_p = β * (quad(intp1,-bound,epstar)[0] + quad(intp2,epstar,bound)[0])
                        const_p = β * (quad(intp1,-bound,epstar, args=(fk, ψ2, b))[0]\
                                       + quad(intp2,epstar,bound, args=(fk, θ2, ψ2, 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*θ2-p*b)**(-ψ2))
                            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 ϵ: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*(Y(ϵ, fk) - b)*g(ϵ)
                        const_qq2 = β * quad(intqq2,epstar,bound, args=(fk, θ2, ψ2, b))[0]
                        qq2l = 0
                        qq2h = ww20
                        diff = 1
                        while diff > 1e-7:
                            qq2 = (qq2l+qq2h)/2
                            rhs = const_qq2/((ww20-qq2*θ2-p*b)**(-ψ2));
                            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:
                            θ1a = θ1
                        else:
                            θ1b = θ1

                    #================
                    # Get consumption
                    #================
                    c10 = ww10 - q*θ1
                    c11 = lambda ϵ: w11(ϵ) + θ1*max(Y(ϵ, fk)-b,0)
                    c20 = ww20 - q*(1-θ1) - p*b
                    c21 = lambda ϵ: w21(ϵ) + (1-θ1)*max(Y(ϵ, fk)-b,0) + min(Y(ϵ, fk),b)


                    #************************************************
                    # Compute the first order conditions for the firm
                    #************************************************
```

```
                    #===========
                    # Equity FOC
                    #===========
                    # Only agent 2's IMRS is relevent
#                     intk1 = lambda ϵ: (w21(ϵ) + Y(ϵ, fk))**(-ψ2)*np.exp(ϵ)*g(ϵ)
#                     intk2 = lambda ϵ: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*np.exp(ϵ)*g(ϵ)
#                     kfoc_num = quad(intk1,-bound,epstar)[0] + quad(intk2,epstar,bound)[0]
                    kfoc_num = quad(intk1,-bound,epstar, args=(fk, ψ2))[0] + quad(intk2,epstar,bound, args=(fk, θ2, ψ2, b))[0]
                    kfoc_denom = (ww20- q*θ2 - p*b)**(-ψ2)
                    kfoc = β*α*A*(k**(α-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 ϵ: (w11(ϵ) + θ1*(Y(ϵ, fk) - b))**(-ψ1)*g(ϵ)
#                 intB2 = lambda ϵ: (w21(ϵ) + θ2*(Y(ϵ, fk) - b) + b)**(-ψ2)*g(ϵ)

#                 bfoc1 = quad(intB1,epstar,bound)[0] / (ww10 - q*θ1)**(-ψ1)
#                 bfoc2 = quad(intB2,epstar,bound)[0] / (ww20 - q*θ2 - p*b)**(-ψ2)

                bfoc1 = quad(intB1,epstar,bound, args=(fk, θ1, ψ1, b))[0] / (ww10 - q*θ1)**(-ψ1)
                bfoc2 = quad(intB2,epstar,bound, args=(fk, θ2, ψ2, b))[0] / (ww20 - q*θ2 - p*b)**(-ψ2)
                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))

            # 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
        θ1ss = θ1


        # 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,θ1ss


    #************************************************************
    # Function: Equity and bond valuations by different agents
    #************************************************************
    def valuations_by_agent(self,
                            c10, c11, c20, c21,
                            k, b):

        # Load parameters
        ψ1 = self.ψ1
        ψ2 = self.ψ2
        α = self.α
        A = self.A
        β = self.β
        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
        θ10 = self.θ10
        θ20 = self.θ20
        w11 = self.w11
        w21 = self.w21
        g = self.g

        # Get functions for IMRS/state price density
        IMRS1 = lambda ϵ: β * (c11(ϵ)/c10)**(-ψ1)*g(ϵ)
        IMRS2 = lambda ϵ: β * (c21(ϵ)/c20)**(-ψ2)*g(ϵ)

        # Production
```

```
        fk = A*(k**α)
        Y = lambda ϵ: np.exp(ϵ)*fk

        # Compute integration threshold
        epstar = np.log(b/fk)

        # Compute equity valuation with agent 1's IMRS
        intQ1 = lambda ϵ: IMRS1(ϵ)*(Y(ϵ) - b)
        Q1 = quad(intQ1, epstar, bound)[0]

        # Compute bond valuation with agent 1's IMRS
        intP1 = lambda ϵ: IMRS1(ϵ)*Y(ϵ)/b
        P1 = quad(intP1, -bound, epstar)[0] + quad(IMRS1, epstar, bound)[0]

        # Compute equity valuation with agent 2's IMRS
        intQ2 = lambda ϵ: IMRS2(ϵ)*(Y(ϵ) - b)
        Q2 = quad(intQ2, epstar, bound)[0]

        # Compute bond valuation with agent 2's IMRS
        intP2 = lambda ϵ: IMRS2(ϵ)*Y(ϵ)/b
        P2 = quad(intP2, -bound, epstar)[0] + quad(IMRS2, epstar, bound)[0]

        return Q1,Q2,P1,P2


    #************************************************************
    # Function: equilibrium valuations for firm, equity, bond
    #************************************************************
    def eq_valuation(self, c10, c11, c20, c21, N=30):

        # load parameters
        ψ1 = self.ψ1
        ψ2 = self.ψ2
        α = self.α
        A = self.A
        β = self.β
        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
        θ10 = self.θ10
        θ20 = self.θ20
        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
                Vgrid[i,j] = V
                Qgrid[i,j] = Q
                Pgrid[i,j] = P

        return kgrid, bgrid, Vgrid, Qgrid, Pgrid
```

38.6. Examples

Below we show some examples computed with the class `BCG_incomplete markets`.

38.6.1. 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,θ1ss = 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(01ss)
```

```
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 = md1.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))
```

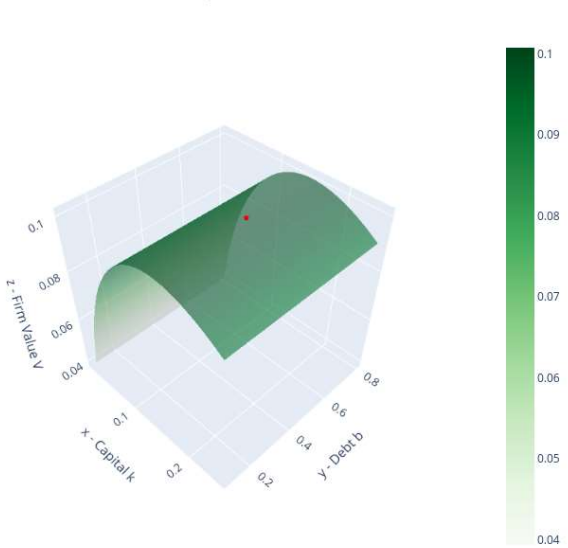
```
Maximum valuation of the firm value in the (k,B) grid: 0.10074
Equilibrium firm value: 0.10083
```

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,
                  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
```



38.6.1.1. 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 (BCG_complete_mkts.html).

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:

$$\begin{aligned} c_0^1 &= w_0^1 + \theta_0^1 V - q\theta^1 - p\xi^1 \\ c_0^2 &= w_0^2 + \theta_0^2 V - q\theta^2 - p\xi^2 \\ c_1^1(\varepsilon) &= w_1^1(\varepsilon) + \theta^1 d^e(k, b; \varepsilon) + \xi^1 \\ c_1^2(\varepsilon) &= w_1^2(\varepsilon) + \theta^2 d^e(k, b; \varepsilon) + \xi^2 \end{aligned}$$

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
    ψ1 = mdl.ψ1
    ψ2 = mdl.ψ2
    α = mdl.α
    A = mdl.A
    β = mdl.β
    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
    θ10 = mdl.θ10
    θ20 = mdl.θ20
    w11 = mdl.w11
    w21 = mdl.w21
    g = mdl.g

    Y = njit(lambda ϵ, fk: np.exp(ϵ)*fk)
    intqq1 = njit(lambda ϵ, fk, θ1, ψ1, ξ1, b: (w11(ϵ) + θ1*(Y(ϵ, fk) - b) + ξ1)**(-ψ1)*(Y(ϵ, fk) - b)*g(ϵ))
    intpp1a = njit(lambda ϵ, fk, ψ1, ξ1, b: (Y(ϵ, fk)/b)*(w11(ϵ) + Y(ϵ, fk)/b*ξ1)**(-ψ1)*g(ϵ))
    intpp1b = njit(lambda ϵ, fk, θ1, ψ1, ξ1, b: (w11(ϵ) + θ1*(Y(ϵ, fk)-b) + ξ1)**(-ψ1)*g(ϵ))
    intpp2a = njit(lambda ϵ, fk, ψ2, ξ2, b: (Y(ϵ, fk)/b)*(w21(ϵ) + Y(ϵ, fk)/b*ξ2)**(-ψ2)*g(ϵ))
    intpp2b = njit(lambda ϵ, fk, θ2, ψ2, ξ2, b: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + ξ2)**(-ψ2)*g(ϵ))
    intqq2 = njit(lambda ϵ, fk, θ2, ψ2, b: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*(Y(ϵ, fk) - b)*g(ϵ))


    # 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 + θ10*V
        ww20 = w20 + θ20*V

        # Production
        fk = A*(k**α)
#         Y = lambda ϵ: np.exp(ϵ)*fk

        # Compute integration threshold
        epstar = np.log(b/fk)


        #****************************************************************
        # 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
        ξ1a = 0
        ξ1b = b/2
        p = 0.3

        while abs(ξ1b - ξ1a) > 0.001:

            ξ1 = (ξ1a + ξ1b) / 2
            θ1a = 0.3
            θ1b = 1

            while abs(θ1b - θ1a) > (0.001/b):

                θ1 = (θ1a + θ1b) / 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, βE[u'(c^{1}_{1})d^{e}(k,B)]
#                 intqq1 = lambda ϵ: (w11(ϵ) + θ1*(Y(ϵ, fk) - b) + ξ1)**(-ψ1)*(Y(ϵ, fk) - b)*g(ϵ)
#                 const_qq1 = β * quad(intqq1,epstar,bound)[0]
                const_qq1 = β * quad(intqq1,epstar,bound, args=(fk, θ1, ψ1, ξ1, 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*θ1-p*ξ1)**(-ψ1));
                    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, βE[u'(c^{1}_{1})d^{b}(k,B)]
#                 intpp1a = lambda ϵ: (Y(ϵ, fk)/b)*(w11(ϵ) + Y(ϵ, fk)/b*ξ1)**(-ψ1)*g(ϵ)
#                 intpp1b = lambda ϵ: (w11(ϵ) + θ1*(Y(ϵ, fk)-b) + ξ1)**(-ψ1)*g(ϵ)
#                 const_pp1 = β * (quad(intpp1a,-bound,epstar)[0] + quad(intpp1b,epstar,bound)[0])
                const_pp1 = β * (quad(intpp1a,-bound,epstar, args=(fk, ψ1, ξ1, b))[0] \
                                 + quad(intpp1b,epstar,bound, args=(fk, θ1, ψ1, ξ1, 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*θ1-pp1*ξ1)**(-ψ1))
                    if (rhs > pp1):
                        pp1l = pp1
                    else:
                        pp1h = pp1
                    diff = abs(pp1l-pp1h)

                #========
                # Agent 2
                #========
                ξ2 = b - ξ1
                θ2 = 1 - θ1
```

```
                # 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, βE[u'(c^{2}_{1})d^{b}(k,B)]
#                 intpp2a = lambda ϵ: (Y(ϵ, fk)/b)*(w21(ϵ) + Y(ϵ, fk)/b*ξ2)**(-ψ2)*g(ϵ)
#                 intpp2b = lambda ϵ: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + ξ2)**(-ψ2)*g(ϵ)
#                 const_pp2 = β * (quad(intpp2a,-bound,epstar)[0] + quad(intpp2b,epstar,bound)[0])
                const_pp2 = β * (quad(intpp2a,-bound,epstar, args=(fk, ψ2, ξ2, b))[0] \
                                 + quad(intpp2b,epstar,bound, args=(fk, θ2, ψ2, ξ2, 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*θ2-pp2*ξ2)**(-ψ2))
                    if (rhs > pp2):
                        pp2l = pp2
                    else:
                        pp2h = pp2
                    diff = abs(pp2l-pp2h)

                # 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 ϵ: (w21(ϵ) + θ2*(Y(ϵ, fk)-b) + b)**(-ψ2)*(Y(ϵ, fk) - b)*g(ϵ)
#                 const_qq2 = β * quad(intqq2,epstar,bound)[0]
                const_qq2 = β * quad(intqq2,epstar,bound, args=(fk, θ2, ψ2, b))[0]
                qq2l = 0
                qq2h = ww20
                diff = 1
                while diff > 1e-7:
                    qq2 = (qq2l+qq2h)/2
                    rhs = const_qq2/((ww20-qq2*θ2-p*ξ2)**(-ψ2));
                    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:
                    θ1a = θ1
                else:
                    θ1b = θ1

                #print(p,q,ξ1,θ1)

            if pp1 > pp2:
                ξ1a = ξ1
            else:
                ξ1b = ξ1


        #================
        # Get consumption
        #================
        c10 = ww10 - q*θ1 - p*ξ1
        c11 = lambda ϵ: w11(ϵ) + θ1*max(Y(ϵ, fk)-b,0) + ξ1*min(Y(ϵ, fk)/b,1)
        c20 = ww20 - q*(1-θ1) - p*(b-ξ1)
        c21 = lambda ϵ: w21(ϵ) + (1-θ1)*max(Y(ϵ, fk)-b,0) + (b-ξ1)*min(Y(ϵ, 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)

    return V,k,b,p,q,c10,c11,c20,c21,ξ1
```

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 $\boldsymbol{K}$ and big $\boldsymbol{B}$.

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 $\boldsymbol{k}$ and $\boldsymbol{b}$.

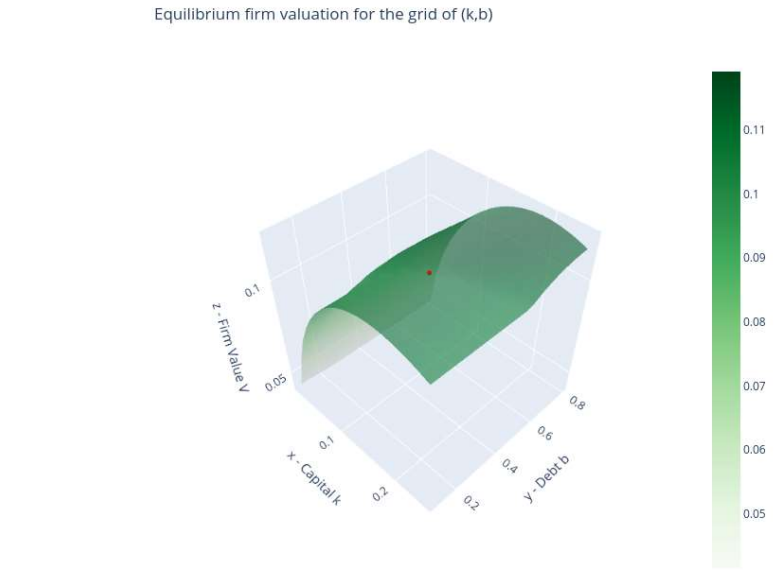
Our hunch is that $\boldsymbol{(k^*, b^{**})}$ is **not** at the top of the firm valuation 3D surface so that the firm is **not** maximizing its value if it chooses $\boldsymbol{k = K = k^*}$ and $\boldsymbol{b = B = b^{**}}$.

That indicates that $\boldsymbol{(k^*, b^{**})}$ is not an equilibrium capital structure for the firm.

We first check the case in which $\boldsymbol{b^{**} = b^* - e}$ where $\boldsymbol{e = 0.1}$:

```
#====================== Experiment 1 ======================#
Ve1,ke1,be1,pe1,qe1,c10e1,c11e1,c20e1,c21e1,ξ1e1 = 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,ξ1e2 = off_eq_check(mdl,
                                                               kss,
                                                               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(ξ1e2))
```

```
Bond holdings of agent 1: 0.039
```

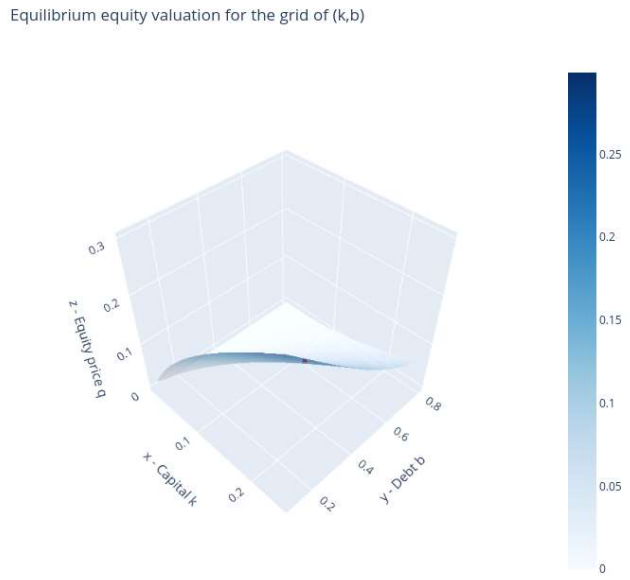
Our two *stability experiments* suggest that the equilibrium capital structure (k^*, b^*) is locally unique even though **at the equilibrium** an individual firm would be willing to deviate from the representative firms' equilibrium debt choice.

These experiments thus refine our discussion of the *qualified* Modigliani-Miller theorem that prevails in this example economy.

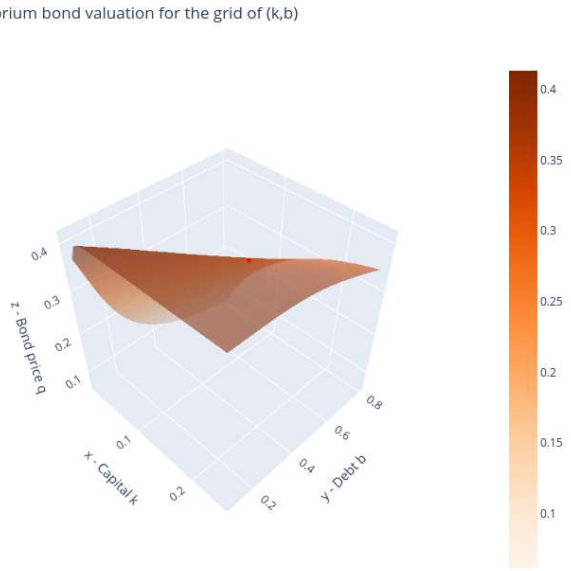
38.6.1.2. Equilibrium equity and bond price functions

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.

```
# 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',
                    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
```



38.6.2. 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 $k(\zeta), b(\zeta)$.

38.6.3. 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.

$$Q^i = \beta \int \frac{u'(C_1^{i,*}(\varepsilon))}{u'(C_0^{i,*})} d^e(k^*, b^*; \varepsilon) g(\varepsilon)\, d\varepsilon$$
$$P^i = \beta \int \frac{u'(C_1^{i,*}(\varepsilon))}{u'(C_0^{i,*})} d^b(k^*, b^*; \varepsilon) g(\varepsilon)\, d\varepsilon$$

The function `valuations_by_agent` is used in calculating these valuations.

```
# 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,θ1ss = 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(θ1ss)

    # 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']
```

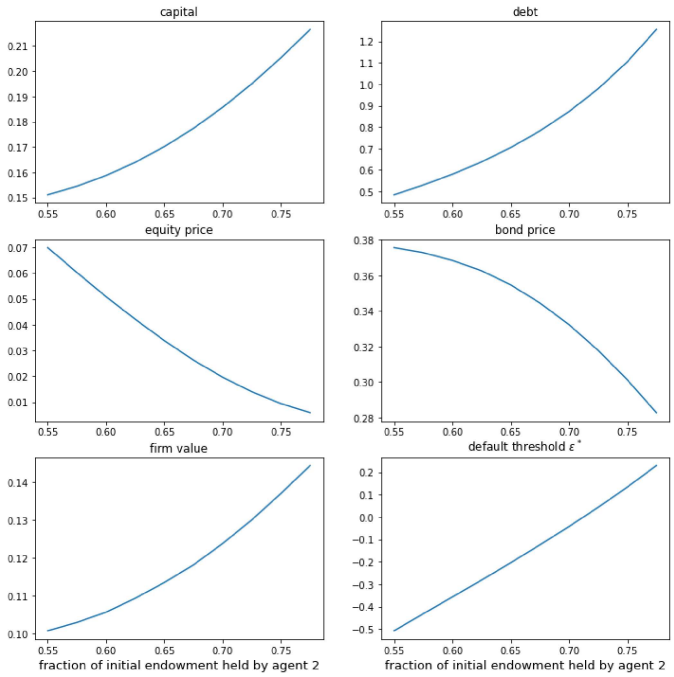
```
k,b,p,q,kfoc,bfoc,epstar,V,V_crit
['0.169', '0.788', '0.330', '0.022', '0.000', '-0.000', '-0.088', '0.062', '0.050']

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
```

```
# 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
α = mdl.α
epslist = []
for i in range(len(wlist)):
    bb = blist[i]
    kk = klist[i]
    eps = np.log(bb/(A*kk**α))
    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()
```



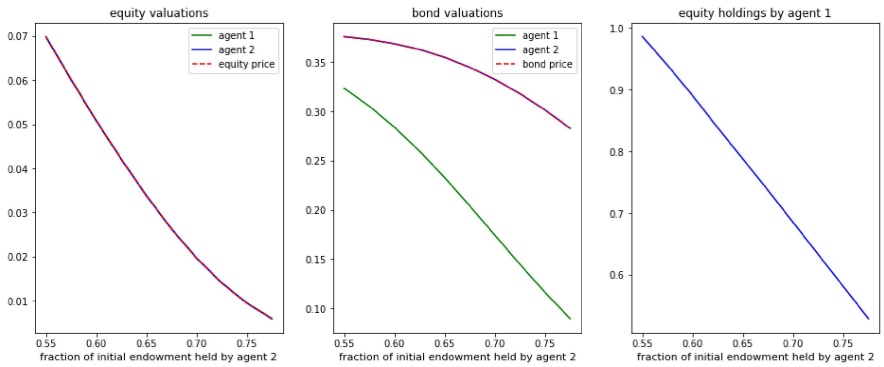
38.7. A picture worth a thousand words

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).

```
# 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.
