Asset Pricing II: The Lucas Asset Pricing Model

Jesse Perla, Thomas J. Sargent and John Stachurski

December 4, 2020

1 Contents

2 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 [1] 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 continous rather than discrete.

3 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.

3.1 Basic Setup

Let's review the set up.

3.1.1 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.

3.1.2 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

3.2 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.

3.2.1 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 .
- 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 equilibrum 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 $c(\pi, y) = \pi(y + p(y)) - \pi'(\pi, y) p(y)$.

3.2.2 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.

3.2.3 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.

3.2.4 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.

3.3 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.

3.3.1 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

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

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.

3.3.2 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 $\alpha := \beta$.

3.4 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.

Some code to implement the iterative computational procedure can be found below:

3.5 Setup

```
In [1]: using InstantiateFromURL
        # optionally add arguments to force installation: instantiate = true,
↪precompile = true
        github_project("QuantEcon/quantecon-notebooks-julia", version = "0.8.0")
```

```
In [2]: using LinearAlgebra, Statistics
        using Distributions, Interpolations, Parameters, Plots, QuantEcon, Random
        gr(fmt = :png);
```

```
In [3]: # model
        function LucasTree(;γ = 2.0,
                            β = 0.95,
                            α = 0.9,
                            σ = 0.1,
                            grid_size = 100)

            ϕ = LogNormal(0.0, σ)
            shocks = rand(ϕ, 500)
```

```
        # build a grid with mass around stationary distribution
        ssd = σ / sqrt(1 - α^2)
        grid_min, grid_max = exp(-4ssd), exp(4ssd)
        grid = range(grid_min, grid_max, length = grid_size)

        # set h(y) = β * int u'(G(y,z)) G(y,z) ⊠(dz)
        h = similar(grid)
        for (i, y) in enumerate(grid)
            h[i] = β * mean((y^α .* shocks).^(1 - γ))
        end

        return (γ = γ, β = β, α = α, σ = σ, ϕ = ϕ, grid = grid, shocks = shocks,
↪h = h)
    end

    # approximate Lucas operator, which returns the updated function Tf on the
↪grid
    function lucas_operator(lt, f)

        # unpack input
        @unpack grid, α, β, h = lt
        z = lt.shocks

        Af = LinearInterpolation(grid, f, extrapolation_bc=Line())

        Tf = [ h[i] + β * mean(Af.(grid[i]^α .* z)) for i in 1:length(grid) ]
        return Tf
    end

    # get equilibrium price for Lucas tree
    function solve_lucas_model(lt;
                               tol = 1e-6,
                               max_iter = 500)

        @unpack grid, γ = lt

        i = 0
        f = zero(grid)  # Initial guess of f
        error = tol + 1

        while (error > tol) && (i < max_iter)
            f_new = lucas_operator(lt, f)
            error = maximum(abs, f_new - f)
            f = f_new
            i += 1
        end

        # p(y) = f(y) * y ^ γ
        price = f .* grid.^γ

        return price
    end
```

```
Out[3]: solve_lucas_model (generic function with 1 method)
```

An example of usage is given in the docstring and repeated here

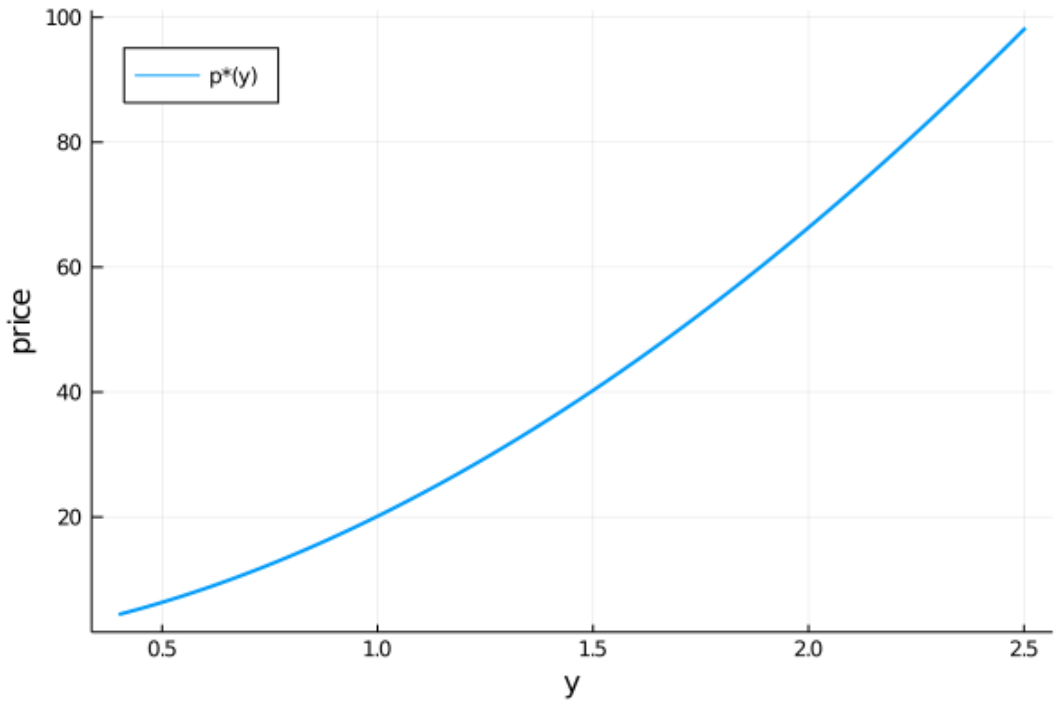
In [4]:
```
Random.seed!(42) # For reproducible results.

tree = LucasTree(γ = 2.0, β = 0.95, α = 0.90, σ = 0.1)
price_vals = solve_lucas_model(tree);
```

Here's the resulting price function

In [5]:
```
plot(tree.grid, price_vals, lw = 2, label = "p*(y)")
plot!(xlabel = "y", ylabel = "price", legend = :topleft)
```

Out[5]:



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

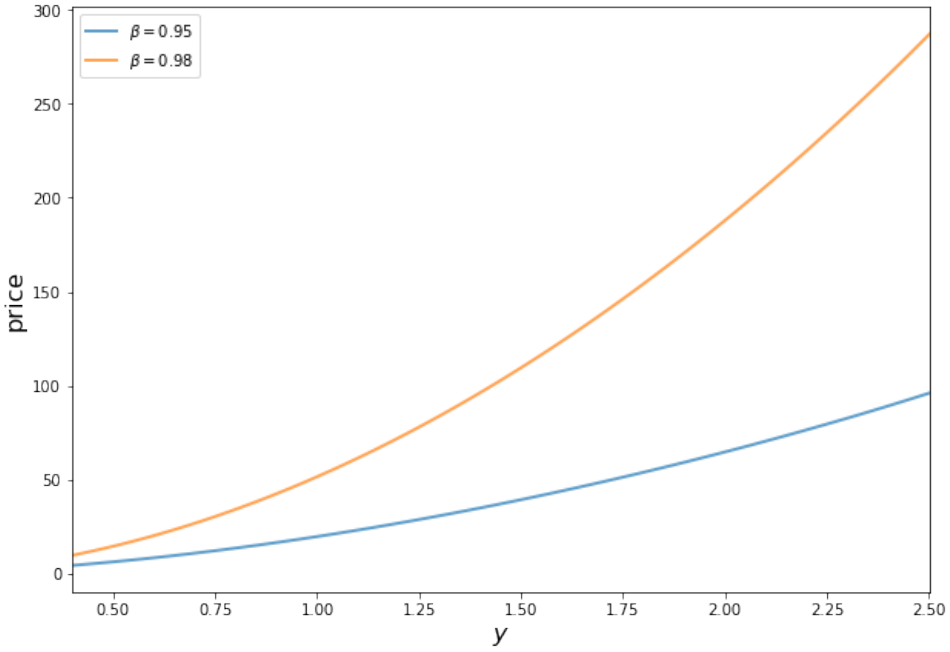
The reason is that a larger current endowment reduces current marginal utility.

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.

4 Exercises

4.1 Exercise 1

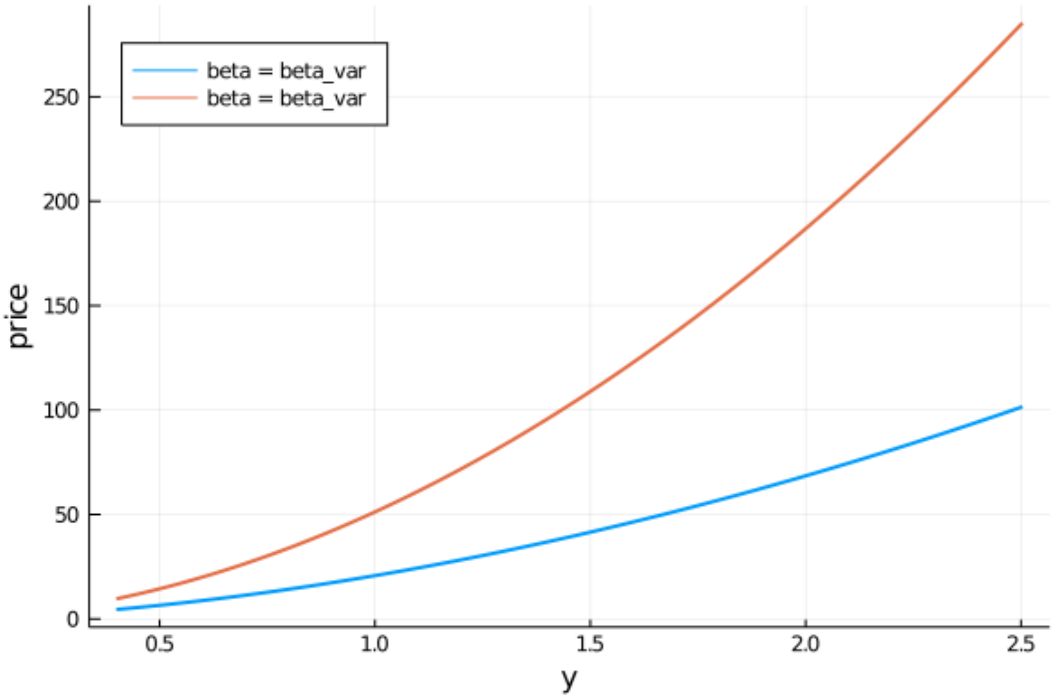
Replicate the figure to show how discount rates affect prices.

5 Solutions

In [6]:
```
plot()
for β in (.95, 0.98)
    tree = LucasTree(;β = β)
    grid = tree.grid
    price_vals = solve_lucas_model(tree)
    plot!(grid, price_vals, lw = 2, label = "beta = beta_var")
end

plot!(xlabel = "y", ylabel = "price", legend = :topleft)
```

Out[6]:



References

[1] Robert E Lucas, Jr. Asset prices in an exchange economy. *Econometrica: Journal of the Econometric Society*, 46(6):1429–1445, 1978.