This document describes the models in the folder `/ConsumptionSaving` as of 6/25/16.

1 ConsIndShockModel.py

Defines consumption-saving models whose agents have CRRA utility over a unitary consumption good, geometric discounting, and who face idiosyncratic shocks to income.

1.1 Perfect Foresight

Consider an agent with CRRA utility over consumption, who discounts future utility at a constant rate per period and has no bequest motive. His problem can be written as:

$$\begin{aligned} V_t(M_t) &= \max_{C_t} \mathrm{u}(C_t) + \beta \not{\mathsf{D}}_{t+1} \mathbb{E}[V_{t+1}(M_{t+1})], \\ A_t &= M_t - C_t, \\ M_{t+1} &= \mathsf{R} A_t + Y_{t+1}, \\ Y_{t+1} &= \Gamma_{t+1} Y_t, \\ \mathrm{u}(C) &= \frac{C^{1-\rho}}{1-\rho}. \end{aligned}$$

The model can be normalized by current income (which is also permanent income in this model) by defining lower case variables as their upper case version divided by Y_t:

$$\begin{aligned} \mathrm{v}_t(m_t) &= \max_{c_t} \mathrm{u}(c_t) + \beta \not{\mathsf{D}}_{t+1} \mathbb{E}[\mathrm{v}_{t+1}(m_{t+1})], \\ a_t &= m_t - c_t, \\ m_{t+1} &= (\mathsf{R}/\Gamma_{t+1}) a_t + 1, \\ \mathrm{u}(c) &= \frac{c^{1-\rho}}{1-\rho}. \end{aligned}$$

An individual agent's model is thus characterized by values of ρ, β, and R along with sequences $\{\Gamma_t\}_{t=1}^T$ and $\{\not{\mathsf{D}}_t\}_{t=1}^T$, with $T = \infty$ possible.

The one period problem for this model is solved by the function `solveConsPerfForesight`, which creates an instance of the class `ConsPerfForesightSolver`. The class `PerfForesightConsumerType` extends `AgentType` to represent agents in this model. The concordance between model variables and their code equivalents is as follows.

Var	Description	Code
ρ	Coefficient of relative risk aversion	`CRRA`
β	Intertemporal discount factor	`DiscFac`
R	Risk free interest factor	`Rfree`
$\not{\mathsf{D}}$	Survival probability	`LivPrb`
Γ	Permanent income growth factor	`PermGroFac`

These are the only five parameters that an instance of `PerfForesightConsumerType` must have in order to use its `solve` method. Note that `LivPrb` and `PermGroFac` are assumed to be time-varying, so they should be input as a list. Each element of the `solution` attribute will be an instance of `ConsumerSolution` with the following attributes:

Var	Description	Code
$\mathrm{c}(\cdot)$	Noramlized consumption function	`cFunc`
$\mathrm{v}(\cdot)$	Normalized value function	`vFunc`
$\mathrm{v}'(\cdot)$	Normalized marginal value function	`vPfunc`
$\underline{m}$	Mininum normalized market resources	`mNrmMin`
h	Normalized human wealth	`hNrm`
$\overline{\kappa}$	Maximum marginal propensity to consume	`MPCmax`
$\underline{\kappa}$	Minimum marginal propensity to consume	`MPCmin`

In the perfect foresight model, the consumption function is linear, so the maximum and minimum MPC are equal. Each of the functions takes normalized market resources m as an argument, and they only defined on the domain $m \geq \underline{m} = -h$.

1.2 Permanent and Transitory Idiosyncratic Shocks

Consider an agent with CRRA utility over consumption, who discounts future utility at a constant rate per period and has no bequest motive. He foresees that he will experience shocks to his income that are fully transitory or fully permanent. Using the normalization above, his problem can be written as:

$$\begin{aligned}
\mathrm{v}_t(m_t) &= \max_{c_t} \mathrm{u}(c_t) + \beta \mathbb{D}_{t+1} \mathbb{E}[\mathrm{v}_{t+1}(m_{t+1})], \\
a_t &= m_t - c_t, \\
a_t &\geq \underline{a}, \\
m_{t+1} &= \mathsf{R}/(\Gamma_{t+1}\psi_{t+1})a_t + \theta_{t+1}, \\
\theta_t \sim F_{\theta t}, & \quad \psi_t \sim F_{\psi t}, \;\; \mathbb{E}[F_{\psi t}] = 1, \\
\mathrm{u}(c) &= \frac{c^{1-\rho}}{1-\rho}.
\end{aligned}$$

That is, this agent is identical to the perfect foresight agent except that his income is subject to permanent (ψ) and transitory (θ) shocks to income, and he might have an artificial borrowing constraint $\underline{a}$.

The one period problem for this model is solved by the function `solveConsIndShock`, which creates an instance of the class `ConsIndShockSolver`. The class `IndShockConsumerType` extends `PerfForesightConsumerType` to represent agents in this model. To construct an instance of this class, several additional parameters must be passed to the constructor. Note that most of these parameters are *indirect* inputs to the consumer's model: they are

used to construct direct inputs to the one period problem. The concordance between the model and code is as follows:

Var	Description	Code
(none)	Minimum of "assets above minimum" grid	`aXtraMin`
(none)	Maximum of "assets above minimum" grid	`aXtraMax`
(none)	Number of points in "assets above minimum" grid	`aXtraCount`
(none)	Additional values for the "assets above minimum" grid	`aXtraExtra`
(none)	Degree of exponential nesting for assets grid	`exp_nest`
N_θ	Number of discrete values in transitory shock distribution	`TranShkCount`
N_ψ	Number of discrete values in permanent shock distribution	`PermShkCount`
σ_θ	Standard deviation of log transitory shocks	`TranShkStd`
σ_ψ	Standard deviation of log permanent shocks	`PermShkStd`
$\mho$	Unemployment probability in working period	`UnempPrb`
$\mho_{ret}$	"Unemployment" probability in retirement period	`UnempPrbRet`
$\underline{\theta}$	Transitory income when unemployed in working period	`IncUnemp`
$\underline{\theta}_{ret}$	Transitory income when "unemployed" in retired period	`IncUnempRet`
τ	Marginal income tax rate	`tax_rate`
T_{ret}	Period of retirement; number of working periods	`T_retire`
$\underline{a}$	Artificial borrowing constraint	`BoroCnstArt`
(none)	Indicator for whether `cFunc` should use cubic splines	`CubicBool`
(none)	Indicator for whether `vFunc` should be computed	`vFuncBool`
T	Total number of (non-terminal) periods in sequence	`T_total`
(none)	Number of agents of this type	`Nagents`

The first five attributes in the table above are used to construct the "assets above minimum" grid `aXtraGrid`, an input for `solveConsIndShock`.[1] The next ten attributes specify an assumed form for the income distribution $(F_{\psi t}, F_{\theta t})$. Both permanent and transitory shocks are lognormally distributed, and with a point mass in the transitory distribution representing unemployment. Further, the sequence of periods is broken into two parts, "working" and "retired" to allow for a different income process in retirement.[2] The attributes `PermShkStd` and `TranShkStd` are thus lists of the (log) standard deviation of shocks period-by-period.

Like the assets grid, the specification of the income process can be changed with little difficulty. No matter what form is used, the relevant direct input to `solveConsIndShock` is `IncomeDstn`, a finite discrete approximation to the true income process. This attribute

[1] In the current configuration, the grid is multi-exponentially spaced given minimum, maximum, number of gridpoints, and degree of exponential nesting (with additional values to force into the grid with `aXtraExtra`). It is simple to replace this grid with another by changing the function `makeAssetsGrid`.

[2] Permanent and transitory shocks are turned off during retirement, other than the possibility of "unemployment", representing (say) a temporary failure of the retirement benefit system.

is specified as a list with three elements: an array of probabilities (that sum to 1), an array of permanent income shocks, and an array of transitory income shocks.

The artificial borrowing constraint imposes a restriction on assets at the end of the period; it can be set to `None` to turn off the constraint (i.e. only the "natural" borrowing constraint will be used). The attributes `CubicBool` and `vFuncBool` should be set to `True` or `False`, as their name implies. The solver can construct a linear or cubic spline interpolation of the consumption function; cubic interpolation is slower but more accurate at any number of gridpoints. The value function is not strictly necessary to compute during solution and carries a computational burden, so it can be turned off with `vFuncBool=False`. The number of agents of this type `Nagents` is irrelevant during solution and is only used during simulation (when *ex-post* heterogeneity emerges within the *ex-ante* homogeneous type).

The `solve` method of `IndShockConsumerType` will populate the `solution` attribute with a list containing instances of `ConsumerSolution`. Each of these instances has all the elements listed above in the perfect foresight section plus the attribute `vPPfunc` (representing $\mathrm{v}''(m)$) if `CubicBool=True`.[3] The problem is solved using the method of endogenous gridpoints, which is explained for this model in section 5.8 of this set of lecture notes.

1.3 Different Interest Rate on Borrowing vs Saving

Consider an agent identical to the "idiosyncratic shocks" model above, except that his interest factor differs depending on whether he borrows or saves on net. His problem is the same as the one above, with a simple addition:

$$\mathsf{R} = \begin{cases} \mathsf{R}_{boro} & \text{if } a_t < 0 \\ \mathsf{R}_{save} & \text{if } a_t > 0 \end{cases}, \qquad \mathsf{R}_{boro} \geq \mathsf{R}_{save}.$$

The one period problem for this model is solved by `solveConsKinkedR`, which creates an instance of `ConsKinkedRsolver`. The class `KinkedRconsumerType` extends `IndShockConsumerType` to represent agents in this model. The attributes required to specify an instance of `KinkedRconsumerType` are the same as `IndShockConsumerType` except that `Rfree` *should not* be included, instead replaced by values of `Rboro` and `Rsave`. The "kinked R" solver is not yet compatible with cubic spline interpolation for `cFunc`; if the `solve` method is run with `CubicBool=True`, it will throw an exception.[4]

The `solve` method of `KinkedRconsumerType` populates the `solution` attribute with a list of `ConsumerSolution` instances, in the same format as the idiosyncratic shocks model. The problem is solved using the method of endogenous gridpoints with *two* copies of $a_t = 0$ in the grid of end-of-period states– one for R_{boro} and the other for R_{save}. This generates the "kinked" portion of the resulting consumption function, where the consumer is unwilling to borrow and insufficiently motivated to save, so he consumes at $c_t = m_t$.

[3] `vFunc` will be a placeholder function of the class `NullFunc` if `vFuncBool=False`.

[4] This is an item that is ripe for development by an outside contributor.

2 ConsPrefShockModel.py

Defines consumption-saving models whose agents have CRRA utility over a unitary consumption good, geometric discounting, who face idiosyncratic shocks to income and to their utility or preferences.

2.1 Multiplicative Shocks to Utility

Consider an agent with a very similar problem to that of the "idiosyncratic shocks" model in the preceding section, except that he receives an iid multiplicative shock to his utility at the beginning of each period, before making the consumption decision. This model can be written in Bellman form as:

$$\begin{aligned}
\mathrm{v}_t(m_t, \eta_t) &= \max_{c_t} \eta \cdot \mathrm{u}(c_t) + \beta \cancel{\mathsf{D}}_{t+1} \mathbb{E}[\mathrm{v}_{t+1}(m_{t+1}, \eta_{t+1})], \\
a_t &= m_t - c_t, \\
a_t &\geq \underline{a}, \\
m_{t+1} &= \mathsf{R}/(\Gamma_{t+1}\psi_{t+1})a_t + \theta_{t+1}, \\
\theta_t \sim F_{\theta t}, & \quad \psi_t \sim F_{\psi t}, \quad \mathbb{E}[F_{\psi t}] = 1, \\
\mathrm{u}(c) &= \frac{c^{1-\rho}}{1-\rho}, \qquad \eta_t \sim F_{\eta t}.
\end{aligned}$$

The one period problem for this model is solved by the function `solveConsPrefShock`, which creates an instance of `ConsPrefShockSolver`. The class `PrefShockConsumerType` is used to represent agents in this model. The attributes required to construct an instance of this class are the same as for `IndShockConsumerType` above, but with three additions:

Var	Description	Code
N_η	Number of discrete points in "body" of preference shock distribution	`PrefShkCount`
N_η^{tail}	Number of discrete points in "tails" of preference shock distribution	`PrefShk_tail_N`
σ_η	Log standard deviation of multiplicative utility shocks	`PrefShkStd`

These attributes are indirect inputs to the problem, used during instantiation to construct the `PrefShkDstn`, an input to `solveConsPrefShock`. The tails of the preference shock distribution matter a great deal for the accuracy of the solution and are underrepresented by the default equiprobable discrete approximation (unless a very large number of points are used). To fix this issue, the attribute `PrefShk_tail_N` specifies the number of points in each "augmented tail" section of the preference shock discrete approximation.[5] The standard deviation of preference shocks might vary by period, so `PrefShkStd` should

[5] See documentation for `HARKutilities.approxLognormal` for more details.

be input as a list. The "preference shock" solver is not yet compatible with cubic spline interpolation for the consumption function and will throw an exception if `CubicBool=True`.

The `solve` method of `PrefShockConsumerType` populates the `solution` attribute with a list of `ConsumerSolution` instaces. These single-period-solution objects have the same attributes as the "idiosyncratic shocks" models above, but the attribute `cFunc` is defined over the space of (m_t, η_t) rather than just m_t. The value function `vFunc` and marginal value `vPfunc`, however, are defined *only* over m_t, as they represent expected (marginal) value *just before* the preference shock η_t is realized:[6]

$$\begin{aligned} \overline{\mathrm{v}}_t(m_t) &= \int_0^\infty \mathrm{v}(m_t, \eta) dF_{\eta t}(\eta), \\ \overline{\mathrm{v}}'_t(m_t) &= \int_0^\infty \mathrm{v}'(m_t, \eta) dF_{\eta t}(\eta). \end{aligned}$$

2.2 Utility Shocks and Different Interest Rates

Consider an agent with idiosyncratic shocks to permanent and transitory income and multiplicative shocks to utility *and* faces a different interest rate on borrowing vs saving. This agent's model is identical to that of the "preference shock" consumer in section 2.1, with the addition of the interest rate rule from the "kinked R" consumer in section 1.3.

The one period problem of this combination model is solved by the function `solveConsKinkyPref`, which creates an instance of `ConsKinkyPrefSolver`. The class `KinkyPrefConsumerType` represents agents in this model. As you will see in `ConsPrefShockModel.py`, there is *very* little new code required to program this model: the solver and consumer classes each inherit from both `KinkedR` and `PrefShock` and only need a trivial constructor function to rectify the differences between the two. This is a good demonstration of the benefit of HARK's object-oriented approach to solution methods: it is sometimes trivial to combine two models to make a new one.

The attributes required to properly construct an instance of `KinkyPrefConsumerType` are the same as for `PrefShockConsumerType` except that (like the "kinked R" parent model) `Rfree` should not be replaced with `Rboro` and `Rsave`. Like both of its parents, `KinkyPref` is not yet compatible with cubic spline interpolation of the consumption function.

[6] Particularly in the case of `vPfunc`, this is the object of interest for solving the preceding period.

3 ConsMarkovModel.py

Defines consumption-saving models with a discrete state that evolves according to an exogenous Markov process.

3.1 Markov States and Idiosyncratic Shocks

Consider an agent with CRRA utility over consumption who geometrically discounts future utility flows and expects to experience transitory and permanent shocks to his income. Moreover, in any given period he finds himself in exactly one of several discrete states; this state evolves from period to period according to a Markov process. The individual's income distribution, permanent income growth rate, and interest factor might vary across states. This agent's problem can be written in Bellman form as:

$$\begin{aligned}
\mathrm{v}_t(m_t, s_t) &= \max_{c_t} \mathrm{u}(c_t) + \beta \not{\mathrm{D}}_{t+1} \mathbb{E}[\mathrm{v}_{t+1}(m_{t+1}, s_{t+1})], \\
a_t &= m_t - c_t, \\
a_t &\geq \underline{a}, \\
m_{t+1} &= \frac{\mathsf{R}(s_{t+1})}{\Gamma_{t+1}(s_{t+1})\psi_{t+1}} \cdot a_t + \theta_{t+1}, \\
\theta_t \sim F_{\theta t}(s_t), & \quad \psi_t \sim F_{\psi t}(s_t), \quad \mathbb{E}[F_{\psi t}(s_t)] = 1, \\
\mathrm{Prob}[s_{t+1} = j | s_t = i] &= \Delta_{ij}, \\
\mathrm{u}(c) &= \frac{c^{1-\rho}}{1-\rho}.
\end{aligned}$$

The Markov matrix is Δ, giving transition probabilities from current state i to future state j. This model is the same as the "idiosyncratic shocks" model of section 2.1 but for the presence of the Markov state s_t, so that the interest factor R, income distribution $(F_{\psi t}, F_{\theta t})$, and permanent income growth factor Γ_{t+1} are all functions of the Markov state, having a value for each state.

The function `solveConsMarkov` solves the one period problem of this model, creating an instance of `ConsMarkovSolver`. The class `MarkovConsumerType` is used to represent agents in this model, extending `IndShockConsumerType`. The attributes required to specify an instance of this class are the same as for `IndShockConsumerType` but for one addition:

Var	Description	Code
Δ	Discrete state transition probability matrix	`MrkvArray`

The attribute `MrkvArray` is a `numpy.array` of size (N_s, N_s) corresponding to the number of discrete states.[7] The attributes `Rfree`, `PermGroFac`, and `IncomeDstn` should be specified

[7] As is, `MrkvArray` is an element of `time_inv`, so the same transition probabilities are used for each period. However, it can be moved to `time_vary` and specified as a list of `arrays` instead.

as lists[8] with N_s elements for each period. Note that `MarkovConsumerType` currently has no method to automatically construct a valid `IncomeDstn`; as seen in the examples in `ConsMarkovModel.py`, the `IncomeDstn` is manually constructed in each case.[9] All other attributes are specified the same as in the "idiosyncratic shocks" model.

When the `solve` method of a `MarkovConsumerType` is invoked, the `solution` attribute is populated with a list of `ConsumerSolution` objects, which each have the same attributes as the "idiosyncratic shocks" model. However, each attribute is now a list (or array) whose elements are *state-conditional* values of that object. For example, in a model with four discrete states, each the `cFunc` attribute of each element of `solution` is a length-4 list whose elements are state-conditional consumption functions (e.g. `cFunc[2]` is the consumption function when $s_t = 2$). The "Markov model" is compatible with cubic spline interpolation for the consumption functions, so `CubicBool=True` will not generate an exception. The problem is solved using the method of endogenous gridpoints, which is moderately more complicated than in the basic "idiosyncratic shocks" model.

4 ConsAggShockModel.py

Defines consumption-saving models with idiosyncratic and aggregate shocks to income.

4.1 Idiosyncratic and Aggregate Shocks to Income

Consider an agent with CRRA preferences over consumption who discounts future utility flows and expects to experience permanent and transitory shocks to his income. He also believes that the market to which he supplies (a fixed amount of) labor will experience *aggregate* permanent and transitory shocks to the effective productivity of labor. The wage rate in the market is the marginal product of labor in the aggregate production function, and the interest factor is one plus the (net) marginal product of capital; assume that the ratio of aggregate capital-to-labor is a sufficient statistic for these marginal products. Further, the agent believes that the capital-to-labor ratio evolves as a function of its current value. This model can be written in Bellman form as:

$$
\begin{aligned}
\mathrm{v}_t(m_t, k_t) &= \max_{c_t} \mathrm{u}(c_t) + \beta \mathbb{D}_{t+1} \mathbb{E}[\mathrm{v}_{t+1}(m_{t+1}, k_{t+1})], \\
a_t &= m_t - c_t, \\
a_t &\geq 0, \\
m_{t+1} &= \frac{\mathsf{R}_{t+1}}{\Gamma_{t+1}\psi_{t+1}\Psi_{t+1}} \cdot a_t + W_{t+1}\theta_{t+1}, \\
\mathsf{R}_{t+1} = \mathbf{R}(k_{t+1}/\Theta_{t+1}), & \quad W_{t+1} = \mathbf{W}(k_{t+1}/\Theta_{t+1}),
\end{aligned}
$$

[8] `PermGroFac` and `Rfree` can be arrays or lists.

[9] Writing a method to supersede `IndShockConsumerType.updateIncomeProcess` for the "Markov model" would be a welcome contribution.

$$
\begin{aligned}
k_{t+1} &= \mathbf{k}(k_t), \\
\theta_t \sim F_{\theta t}, & \quad \psi_t \sim F_{\psi t}, \;\; \mathbb{E}[F_{\psi t}] = 1, \\
\Theta_t \sim F_{\Theta}, & \quad \Psi_t \sim F_{\Psi}, \;\; \mathbb{E}[F_{\Psi}] = \mathbb{E}[F_{\Theta}] = 1, \\
\mathrm{u}(c) &= \frac{c^{1-\rho}}{1-\rho}.
\end{aligned}
$$

The objects $\mathbf{R}(\cdot)$ and $\mathbf{W}(\cdot)$, are functions of the (effective) capital-to-labor ratio that yield the (net) interest factor and wage rate respectively. As noted above, these are determined by the aggregate production function and the degree of capital depreciation. The $\mathtt{k}(\cdot)$ function represents the agent's beliefs about the evolution of the capital-to-labor ratio k_t. As with idiosyncratic shocks, there is an aggregate shock process (F_{Θ}, F_{Ψ}).

The one period problem of this model is solved by the function `solveConsAggShock`, the default value of `solveOnePeriod` for `AggShockConsumerType`. The attributes required to specify an instance of this class are listed in the concordance below.

Var	Description	Code
ρ	Coefficient of relative risk aversion	`CRRA`
β	Intertemporal discount factor	`DiscFac`
$\not\mathbb{D}$	Survival probability	`LivPrb`
Γ	Permanent income growth factor	`PermGroFac`
(none)	Minimum of "assets above minimum" grid	`aXtraMin`
(none)	Maximum of "assets above minimum" grid	`aXtraMax`
(none)	Number of points in "assets above minimum" grid	`aXtraCount`
(none)	Additional values for the "assets above minimum" grid	`aXtraExtra`
(none)	Degree of exponential nesting for assets grid	`exp_nest`
$\{\hat{k}\}$	Array of scaling factors for capital ratio (around SS)	`kGridBase`
N_{θ}	Number of discrete values in transitory shock distribution	`TranShkCount`
N_{ψ}	Number of discrete values in permanent shock distribution	`PermShkCount`
σ_{θ}	Standard deviation of log transitory shocks	`TranShkStd`
σ_{ψ}	Standard deviation of log permanent shocks	`PermShkStd`
$\mathcal{U}$	Unemployment probability in working period	`UnempPrb`
$\mathcal{U}_{ret}$	"Unemployment" probability in retirement period	`UnempPrbRet`
$\underline{\theta}$	Transitory income when unemployed in working period	`IncUnemp`
$\underline{\theta}_{ret}$	Transitory income when "unemployed" in retired period	`IncUnempRet`
τ	Marginal income tax rate	`tax_rate`
T_{ret}	Period of retirement; number of working periods	`T_retire`
T	Total number of (non-terminal) periods in sequence	`T_total`
(none)	Number of agents of this type	`Nagents`

This list is very similar to the one for `IndShockConsumerType`, but several attributes have been removed: `Rfree` is endogenous here, while `CubicBool`, `vFuncBool`, and `BoroCnstArt`

are not yet supported in the "aggregate shocks" model. The only new attribute is `kGridBase`, an array of scaling factors for the (perfect foresight equivalent) steady state capital ratio; it is used to construct `kGrid`, a direct input for `solveConsAggShock`.

A new `AggShockConsumerType` with these attributes is not yet ready to solve its micro model, as it lacks several features. After creating a valid `CobbDouglasEconomy` instance (see section 4.2), the agent type must get "macro" level objects from this by invoking its `getEconomyData` method with the `CobbDouglasEconomy` as the input. This gives the agent type its interest, wage, and next-capital-ratio functions as the attributes `Rfunc`, `Wfunc`, and `kNextFunc`, its capital ratio grid `kGrid`, and reformats the `IncomeDstn` attribute as a discrete joint distribution across all four types of shocks.[10]

After obtaining "macro"-level inputs to its model, an `AggShockConsumerType`'s `solve` method will populate the `solution` attribute with a list of `ConsumerSolution` instances. Unlike the models with only idiosyncratic shocks, the one-period-solution objects have only two attributes, `cFunc` and `vPfunc`; both of these functions are defined over the space of (m_t, k_t). The model is solved using the method of endogenous gridpoints, following Kiichi Tokuoka's Mathematica code for the "cstwMPC" project.

4.2 Cobb-Douglas Economy

A model with "aggregate shocks" only makes sense if there is some market-level object that experiences these shocks. The `CobbDouglasEconomy` class extends `Market` to represent an economy with a Cobb-Douglas production function over aggregate capital and aggregate labor[11] and permanent and transitory shocks to labor productivity. The basic model for the Cobb-Douglas economy is:

$$\begin{aligned} Y &= K^{\alpha}L^{1-\alpha}, \qquad k \equiv K/L, \\ W = \frac{\partial Y}{\partial L} &= (1-\alpha)K^{\alpha}L^{-\alpha} = (1-\alpha)k^{\alpha}, \\ \mathsf{r} = \frac{\partial Y}{\partial K} &= \alpha K^{\alpha-1}L^{1-\alpha} = \alpha k^{\alpha-1}, \\ \mathsf{R} &= 1 + \mathsf{r} - \delta. \end{aligned}$$

A new instance of `CobbDouglasEconomy` must have attributes listed in the table below. The constructor for the class uses these attributes to calculate the perfect foresight steady state of the capital-to-labor ratio[12], wage rate, and interest rate; the interest and wage

[10] As of the beta release, this method is only compatible with one period infinite horizon micro models, but this can be fixed with minimal difficulty.

[11] As the microeconomic model in section 4.1 assumes a fixed per capita labor supply, aggregate labor for this class is assumed constant and disappears into the background. The HARK team welcomes contributions that extend both the micro and macro models to account for endogenous labor supply.

[12] If the economy were populated with perfect foresight agents with preferences given by β^{PF} and ρ^{PF}, the steady state level of capital is where $k_{t+1} = k_t$ (when aggregate shocks are turned off as well).

functions; a discretization of the aggregate shock process, and an initial guess of the next-capital-ratio function. Following Krusell and Smith (1998), we assume that the log of next period's capital ratio is a linear function of the log of this period's capital ratio.

Var	Description	Code
α	Capital's share of output	`CapShare`
δ	Capital depreciation rate	`DeprFac`
σ_Ψ	Standard deviation of log permanent aggregate shocks	`PermShkAggStd`
σ_Θ	Standard deviation of log transitory aggregate shocks	`TranShkAggStd`
N_Ψ	Number of discrete values in permanent agg shock distribution	`PermShkAggCount`
N_Θ	Number of discrete values in transitory agg shock distribution	`TranShkAggCount`
ρ^{PF}	Perfect foresight coefficient of relative risk aversion	`CRRAPF`
β^{PF}	Perfect foresight intertemporal discount factor	`DiscFacPF`

After a well-formed `CobbDouglasEconomy` has been created, its `agents` attribute can be populated with one or more instances of `AggShockConsumerType` (who have taken "macro" level information from the `CobbDouglasEconomy`). A history of aggregate shocks can be created by invoking the `makeAggShkHist` method. If each element of `AggShockConsumerType` has run its `makeIncShkHist` method to create a history of idiosyncratic income shocks (for many agents in each type), then the `CobbDouglasEconomy` can invoke its `solve` method. This will search for a general equilibrium of the model, defined as a "dynamic rule" for the capital ratio $\mathbf{k}(k_t)$ that is *consistent*: when agents believe this $\mathbf{k}(k_t)$ in their microeconomic problem, and the model is simulated for many periods, the resulting history of the capital ratio is consistent with that same dynamic rule.

In the `Market` framework, the `millRule` for `CobbDouglasEconomy` gathers each agent's end-of-period normalized assets a_t and permanent income p_t. It aggregates wealth across all consumers into total capital, which it transforms into the capital-to-labor ratio. It then uses the next aggregate shock values to calculate R_t and W_t, which are distributed back to the consumers along with the aggregate shocks and new capital ratio, so that they can simulated another period.

After generating a history of several thousand periods, the `CobbDouglasEconomy` can calculate a new dynamic rule for the capital ratio with its `calcDynamics` method. The dynamics calculator simply throws out the first 200 periods of the history and runs a one-period-lag autoregression on the log capital ratio. This generates a new function for `kNextFunc`, which is distributed to the consumer types in `agents` to re-solve their micro models. This process continues until successive `kNextFunc`s are sufficiently close to consider the process converged (as determined by the `tolerance` attribute).

5 TractableBufferStockModel.py

Defines the "tractable buffer stock" model from Chris Carroll's lecture notes.

5.1 Tractable Buffer Stock

Consider a consumer with CRRA utility who faces only a single, very specific risk: that he will become permanently unemployed and receive no income until the end of time. Otherwise, he faces an infinite horizon problem with a steady stream of income that grows by a fixed factor each period, and earns a constant rate of return on assets retained betweed periods. His model when still employed can be written in Bellman form as:[13]

$$\begin{aligned} \mathrm{v}^e(m_t) &= \max_{c_t} \mathrm{u}(c_t) + \beta \left((1-\mho)\mathrm{v}^e(m^e_{t+1}) + \mho \mathrm{v}^u(m^u_{t+1}) \right) \\ a_t &= m_t - c_t \\ m^e_{t+1} &= (\mathsf{R}/\widehat{\Gamma})a_t + 1, \qquad \widehat{\Gamma} = \Gamma/(1-\mho) \\ m^u_{t+1} &= (\mathsf{R}/\widehat{\Gamma})a_t. \end{aligned}$$

His model while unemployed is simply:

$$\begin{aligned} \mathrm{v}^u(m_t) &= \max_{c_t} \mathrm{u}(c_t) + \beta \mathrm{v}^u(m^u_{t+1}) \\ a_t &= m_t - c_t \\ m^u_{t+1} &= (\mathsf{R}/\widehat{\Gamma})a_t. \end{aligned}$$

This model is solved by the class `TractableConsumerType` when its `solve()` method is invoked. An instance of this class is specified by the five parameters in the table below:

Var	Description	Code
ρ	Coefficient of relative risk aversion	`CRRA`
β	Intertemporal discount factor	`DiscFac`
R	Interest factor on assets	`Rfree`
Γ	Permanent income growth factor	`PermGroFac`
$\mho$	Probability of becoming unemployed	`UnempPrb`

Unlike other models in HARK, tractable buffer stock is not solved by backward induction beginning from an initial guess of the solution. Because of the very specific form of risk faced by the agent, it is possible to calculate[14] analytical values of the steady state (m_t, c_t) and to find several derivatives of the consumption function at this point (i.e. the MPC, MMPC, etc). Further, the Euler and transition equations can be inverted to yield (m_{t-1}, c_{t-1}) as a function of (m_t, c_t) conditional on being employed in both periods. Beginning from a

[13] For technical / teaching reasons, permanent income growth while employed is "risk compensated" so that human wealth does not vary with the unemployment probability.

[14] As long as the consumer is both "return impatient" and "growth impatient", else there is no steady state or no solution at all.

small perturbation along a Taylor approximation of the consumption function around the steady state, the solution method generates a sequence of "stable arm points" along the consumption function. After reaching specified bounds (and appending the lower bound at $(0, 0)$), the (employed) consumption function is constructed as a cubic spline interpolation.

After running the `solve` method, the `solution` attribute of a `TractableConsumerType` will have a list with a single instance of `TractableConsumerSolution`. This object has the following attributes:

Var	Description	Code
$\{m_t\}$	List of market resources values on the stable arm	`mNrm_list`
$\{c_t\}$	List of consumption values on the stable arm	`cNrm_list`
$\{\kappa_t\}$	List of MPCs at points on the stable arm	`MPC_list`
$\mathrm{c}^e(m_t)$	Consumption function when employed	`cFunc`
$\mathrm{c}^u(m_t)$	Consumption function when unemployed	`cFunc_U`
(none)	Number of stable arm points included	`PointCount`

5.2 Tractable Buffer Stock as Markov

The tractable buffer stock model can also be solved by the standard backward induction approach if it is framed in terms of the Markov model in section 3. There are two discrete state, *employed* and *unemployed*; transition probabilities from the former are $(1-\mathcal{U}, \mathcal{U})$ and the latter is an absorbing state. The interest factor and permanent income growth rate are identical in the two states, and both have degenerate income distributions: $\psi_e = \theta_e = 1$, while $\psi_u = 1$ and $\theta_u = 0$ for sure. The model takes about 300 times longer to solve using the "Markov formulation" as the backshooting method, yielding a nearly identical solution.