Deciphering the Mathematica Code for "MicroDSOP"[1]

Weifeng Wu

July 4, 2018

Contents

[1] This deciphering is based on the archive file dated August 19, 2012.

1 Intro

1.1 Simple Organization Principles

Here are several rules to streamline the `Mathematica` code:

- There are three big parts: `2period`, `multiperiod` and `multicontrol`.
- For each main file like `2period.m`, there is a corresponding file with the post-fix "`!plot`", which is to call the associated file (without the post-fix) and plot the associated figures.
- For each main file, it will
  - load a bunch of files with `setup` as prefix, which set up the environment for solution;
  - load a file with `prepare` as prefix, which defines the `SolveAnotherPeriod` function;
  - run the `SolveAnotherPeriod` function.
- For each `prepare` file, for most parts it will do three major things:
  - define a function called `functions_stable`, i.e. the Gothic V functions that take end-of-period asset as an input and produces the end-of-period value and its derivatives;
  - define a function called `SolveAnotherPeriod` which constructs the grid of cash-on-hand and the corresponding grid of optimal consumptions, and interpolates along the grid pairs;
  - derive the consumption function from the interpolated functions.

1.2 Solution Progression Map

The central problem we are solving is the optimal consumption function at period $T-1$. The above statement about `prepare` is a slight simplification. In fact the various ways we have tried to find the solution, differ on the number of necessary steps.

- Two-step (`2period`): (1) to construct the end of period expected value function $\mathfrak{v}_t(a_t)$ from the value function next period $\mathrm{v}_{t+1}(m_{t+1})$; (2) to find the optimal consumption function at time t, by calling the direct brute-force `FindMaximum` (i.e. for each and every possible cash-on-hand, we need to call the time-consuming `FindMaximum` routine).
- Three-step using exogenous grid points

  - `2periodInt`: the second step above is decomposed to (2) for a number of grid points, to find the optimal consumption grid which involves the direct brute-force `FindMaximum`, and (3) for any arbitrary cash-on-hand value, the corresponding consumption is derived from using the interpolated function, instead from the brute-force `FindMaximum`.
  - `2periodIntExp`: we rely on an interpolated end-of-period value function in step (2) above.
  - `2periodIntExpFOC`: we rely on an interpolated end-of-period marginal value function in step (2) above, and instead of using `FindMaximum`, we use the brute-force `FindRoot` which is not as slow.

- Three-step using endogenous grid points
  - `2periodIntExpFOCInv`: in solving for the optimal consumption grids, we no longer start with a given grid of cash-on-hand and look for the corresponding consumption grid. Instead, we start with a given grid of end-of-period asset, compute their end-of-period marginal value, derive the corresponding consumption and back out the associated cash-on-hand grid. Everything done here is all algebraic calculation, without any involvement on numeric optimizer or root finding. This greatly improves the computational efficiency.
  - `2periodIntExpFOCInvEEE`: we slightly refine the given grid of end-of-period asset, so as to improve the precision of the interpolation-based approximation.

- Five-step (`2periodIntExpFOCInvPesReaOpt`): when we apply the method of moments, after finding the optimal cash-on-hand and consumption grids in `AddNewPeriodToSolvedLifeDates`, we will add a transformation, i.e. defining two grids (μ and χ), which are where the interpolation actually takes place in `AddNewPeriodToSolvedLifeDatesPesReaOpt`. Due to this transformation, we need to derive the χ function from the interpolation and then further back out the optimal consumption function from the χ function.

- Seven-step (`2PeriodIntExpFOCInvPesReaOptTighterUpBd`): when we add the tighter upper bound, we will have a point below which we will utilize the tighter upper bound in defining the χ grid; and correspondingly we will have two χ interpolation functions, and our final reconciled consumption function will be actually be a piecewise function defined on three segments of cash-on-hand.

- Eight-step (`2PeriodIntExpFOCInvPesReaOptTighterUpBdCon`): when we have a liquidity constraint, what we first do is to derive the consumption function assuming the constraint is nonexistent for this period, and then apply the constraint manually in an ad hoc way.

1.3 Setup

The core file that executes everything is called `doAll.nb`[2]; and our deciphering effort will move along the line in this file.

At the beginning it clears everything in the memory, and sets the present working directory as where the `doAll.nb` file is located. Then it loads various `setup` files:

- `setup_workspace.m` defines a function that does the discrete approximation to a mean-one lognormal distribution; sets options for plotting figures; and defines a function that exports the figures to a given location.
- `setup_params.m` defines values for various parameters.
- `setup_params_2period.m` revises the value for the standard error of the log-normal distribution from 0.1 to 1.0, which the transitory income shock is assumed to follow. This unrealistic exaggeration is used to make the following illustration more vivid.
- `setup_shocks.m` invokes the previously-defined discrete approximation function and generates the discrete points with associated probabilities, which approximates the lognormal-distributed shock to the transitory income.
- `setup_shocks_Unem.m` adjusts the discretized transitory shock to incorporate the scenario of unemployment risk $\wp > 0$ and benefit $\underline{\theta} > 0$: suppose $\log \theta_s \sim \mathcal{N}(-\sigma_\theta^2/2, \sigma_\theta^2)$, then taking into account of the unemployment scenario, we assume the new transitory shock is

$$\epsilon_s = \begin{cases} \underline{\theta}_s & \text{with probability } \wp_s > 0 \\ \underline{\theta}_s + \theta_s \frac{1-\underline{\theta}_s}{1-\wp_s} & \text{with probability } (1-\wp_s) \end{cases} \tag{1}$$

- `setup_grids.m` constructs the grids for the cash-on-hand m_t (in case of exogenous grid points) and end-of-period asset a_t (in case of endogenous grid points).
- `setup_lastperiod.m` initializes various lists by defining their elements in the last period of life. In particular, the first element in aLowerBoundLife is assigned to be 0, implying a liquidity constraint in time T, and we set the first element of `ConstrainedLife`[3] as `"Yes"`. It is worthy explaining several human-wealth-related terms:
  - $\mathfrak{h}\text{ExpLife}_t$ is the human wealth at the end of time t, assuming every period onwards the labor income shock will be at its expected level (i.e. $y\text{Exp}_s$);
  - $\mathfrak{h}\text{MinLife}_t$ is the human wealth at the end of time t, assuming every period onwards the individual will be unemployed and receive the unemployment benefit only (i.e. $y\text{Min}_s$);

[2] There is a corresponding `doAll.m`, which contains the code only, not the output.

[3] This term, along with those human-wealth-related terms with a `Borrowable` or `Accessible` keyword, will only be useful when we proceed to the problem with an artificial liquidity constraint.

- $\hbar\text{BorrowableLife}_t$ is the human wealth at the end of time t that can be borrowed against (in the natural liquidity constraint, we have this term equal to $\hbar$MinLife, but in other cases, it will be smaller than $\hbar$MinLife due to the can-not-die-in-debt condition);
- $\hbar\text{AccessibleLife}_t$ is the human wealth at the end of time t that can be borrowed for a consumer who is not constrained at period t only (s/he will be subject to the same constraint for period $t+1$ onwards just as in $\hbar\text{BorrowableLife}_t$)[4];
- $\blacktriangle\hbar\text{MinLife}_t$ is the difference between $\hbar\text{ExpLife}_t$ and $\hbar\text{MinLife}_t$;
- $\blacktriangle\hbar\text{BorrowableLife}_t$ is the difference between $\hbar\text{ExpLife}_t$ and $\hbar\text{BorrowableLife}_t$;
- $\blacktriangle\hbar\text{AccessibleLife}_t$ is the difference between $\hbar\text{ExpLife}_t$ and $\hbar\text{AccessibleLife}_t$;
- $y\text{ExpLife}_t$ is the collection of expected level of income since period t;
- $y\text{MinLife}_t$ is the collection of minimum level of income since period t;
- $y\text{BorrowableLife}_t$ is the period-by-period collection of borrowable level of income since period t;
- $y\text{AccessibeLife}_t$ is the period-by-period collection of accessible level of income (which is the maximum of $y\text{MinLife}_t$ and $y\text{BorrowableLife}_t$) since period t;
- hExpLife_t is the human wealth at the beginning of time t, assuming every period onwards including time t, the labor income shock will be at its expected level;
- hMinLife_t is the human wealth at the beginning of time t, assuming every period onwards including time t, the labor income shock will be at its lowest possible level, i.e. the unemployment benefit.
- hBorrowableLife_t is the human wealth that can be borrowed against[5] at the beginning of time t.

- `setup_definitions.m`, which defines various terms:

$$\begin{aligned} \lambda\text{Sup} &= (R\beta)^{1/\rho}/R && (2)\\ \kappa\text{Inf} &= 1-\lambda\text{Sup} && (3)\\ \text{Þ} &= (R\beta)^{1/\rho} && (4)\\ \text{Þ}_R &= \text{Þ}/R && (5) \end{aligned}$$

  and various functions:

$$u(c) = c^{1-\rho}/(1-\rho) \quad (6)$$

[4] If there is no constraint at period t, then it is the same as $\hbar\text{BorrowableLife}_t$; and if there is, then it differs from $\hbar\text{BorrowableLife}_t$ only by $y\text{Min}_{t+1} - y\text{Borrowable}_{t+1}$ discounted back to period t.

[5] One might wonder if one should also define hAccessibleLife_t, and the answer is no. Basically we define the beginning of period human wealth, as an aid to derive the end of period human wealth one period before. However $\hbar\text{AccessibleLife}_t$ is derived from $\hbar\text{BorrowableLife}_{t+1}$ and so there is no need to do so. Even if we define it, it is not clear what it means.

$$
\begin{aligned}
\mathrm{u}'(c) &= \begin{cases} c^{-\rho} & \forall c > 0 \\ \infty & \forall c \leq 0 \end{cases} && (7)\\
\mathrm{u}''(c) &= -\rho c^{-\rho-1} && (8)\\
n(z) &= (z(1-\rho))^{1/(1-\rho)} && (9)\\
\mathrm{nP}(z) &= z^{-1/\rho} && (10)\\
\mathrm{nPP}(z) &= (-z/\rho)^{-1/(\rho+1)} && (11)
\end{aligned}
$$

- `setup_PerfectForesightSolution.m` defines the consumption functions in the perfect foresight scenario, where there do exist analytical solutions[6]:

$$
\begin{aligned}
\mathrm{cOpt}_t(m) &= \kappa\mathrm{Min}(m + \mathfrak{h}\mathrm{ExpLife}_t) && (18)\\
\mathrm{cPes}_t(m) &= \kappa\mathrm{Min}(m + \mathfrak{h}\mathrm{MinLife}_t) && (19)\\
\kappa\mathrm{Opt}_t(m) &= \kappa\mathrm{Min}_t && (20)\\
\mathfrak{c}\mathrm{Opt}_t(a) &= \mathrm{cOpt}_t(a + \mathfrak{h}\mathrm{ExpLife}_t)/(1 - \kappa\mathrm{Min}_t) && (21)\\
\mathfrak{c}\mathrm{Opt}'_t(a) &= \kappa\mathrm{Min}_t/(1 - \kappa\mathrm{Min}_t) && (22)
\end{aligned}
$$

  and value function:

$$
\begin{aligned}
\mathrm{vOpt}_t(m) &= u(\mathrm{cOpt}_t(m))\mathrm{vSum}_t && (23)\\
\mathrm{vPes}_t(m) &= u(\mathrm{cPes}_t(m))\mathrm{vSum}_t && (24)\\
\mathrm{vOpt}'_t(m) &= \mathrm{u}'(\mathrm{cOpt}_t(m))\kappa\mathrm{Min}_t\mathrm{vSum}_t && (25)\\
\mathrm{vOpt}''_t(m) &= \mathrm{u}''(\mathrm{cOpt}_t(m))(\kappa\mathrm{Min}_t)^2\mathrm{vSum}_t && (26)\\
\mathfrak{v}\mathrm{Opt}_t(a) &= u(\mathfrak{c}\mathrm{Opt}_t(a))(\mathrm{vSum}_t - 1) && (27)\\
\mathfrak{v}\mathrm{Opt}'_t(a) &= \mathrm{u}'(\mathfrak{c}\mathrm{Opt}_t(a))\mathfrak{c}\mathrm{Opt}'_t(a)(\mathrm{vSum}_t - 1) && (28)\\
\mathfrak{v}\mathrm{Opt}''_t(a) &= \mathrm{u}''(\mathfrak{c}\mathrm{Opt}_t(a))(\mathfrak{c}\mathrm{Opt}'_t(a))^2(\mathrm{vSum}_t - 1) && (29)
\end{aligned}
$$

- `setup_everything.m` loads all the above `setup` files.

1.4 `AddNewPeriodToParamLifeDates`

Suppose we have solved the problem at period $t+1$, and now we proceed one period backwards. Before we could actually solve anything, we know there is a not-so-trivial

[6] The derivation for $\mathfrak{c}\mathrm{Opt}_t(a)$ is as follows:

$$
\begin{aligned}
a_t(m) &= m - \mathrm{cOpt}_t(m) && (12)\\
&= (1 - \kappa\mathrm{Min}_t)m - \kappa\mathrm{Min}_t\mathfrak{h}\mathrm{ExpLife}_t && (13)\\
m_t(a) &= \frac{a_t + \kappa\mathrm{Min}_t\mathfrak{h}\mathrm{ExpLife}_t}{1 - \kappa\mathrm{Min}_t} && (14)\\
\mathfrak{c}\mathrm{Opt}_t(a) &= \mathrm{cOpt}_t(m_t(a)) && (15)\\
&= \kappa\mathrm{Min}_t\big(\frac{a + \kappa\mathrm{Min}_t\mathfrak{h}\mathrm{ExpLife}_t}{1 - \kappa\mathrm{Min}_t} + \mathfrak{h}\mathrm{ExpLife}_t\big) && (16)\\
&= \kappa\mathrm{Min}_t(a + \mathfrak{h}\mathrm{ExpLife}_t)/(1 - \kappa\mathrm{Min}_t) && (17)
\end{aligned}
$$

routine[7] on between-period parameter updating, to accommodate the potential time-varying parameter values like in a typical life-cycle framework. This is accomplished by a function in `AddNewPeriodToParamLifeDates.m`: its basic role is to augment various lists as defined in `setup_lastperiod.m` one period ahead.

The most puzzling parts are the liquidity constraints and several human-wealth-related terms. Basically we solve the problem backwards, and at the end of period t we first need to identify whether this period is liquidity constrained (i.e. whether the consumer is able to borrow against next period's minimally guaranteed income[8]): yes if next period's yBorrowable is smaller than next period's yMin, and no if otherwise.

Independently whether there is a constraint or not this period, we need to first solve the unconstrained problem, which requires us to define the maximum amount that is accessible to an unconstrained consumer at the beginning of the period:

- if there is no constraint this period (i.e. Last[yBorrowableLife] $\geq$ Last[yMinLife]), then the maximum accessible is defined on yBorrowable, and we can simply solve the problem knowing comfortably that Last[yBorrowableLife] can be borrowed against.

- if there is a constraint this period, (i.e. Last[yBorrowableLife] $<$ Last[yMinLife]), then the maximum amount that is accessible to the unconstrained consumer is defined on yMin. And we can then apply the constraint to the unconstrained solution in an ad hoc way.

Hence we can define a list called yAccessibeLife that returns the maximum of Last[yBorrowableLife] and Last[yMinLife], and $\mathfrak{h}$AccessibleLife that collects the amount of human wealth accessible to an unconstrained consumer at the beginning of this period. And only after this can we identify the lowest possible value for the end-of-period asset (or equivalently for the beginning-of-period cash): effectively it is the negation of $\mathfrak{h}$AccessibleLife.

With this in-mind, we can update the end-of-period human wealth terms:

$$\mathfrak{h}\text{ExpLife}_t = \sum_{s=t+1}^{T} \left(\prod_{\tau=t+1}^{s} \Gamma_\tau / R_\tau \right) y\text{Exp}_s \tag{30}$$

$$\mathfrak{h}\text{BorrowableLife}_t = \sum_{s=t+1}^{T} \left(\prod_{\tau=t+1}^{s} \Gamma_\tau / R_\tau \right) y\text{Borrowable}_s \tag{31}$$

[7] This is one of the core files, as it updates the various seemingly confusing human wealth terms that are critical for thinking about the liquidity constrained scenarios.

[8] Here we do not consider the time-varying yBorrowable, but it is possible that $y\text{Borrowable}_t > y\text{Min}_t$ for some periods of life and otherwise for some other periods. For example, suppose a person lives for two periods, with $y\text{Min}_1 = 10$ and $y\text{Min}_2 = 20$. If we have $y\text{Borrowable}_1 = 15$, $y\text{Borrowable}_2 = 10$, then in the second period he is constrained but in the first period he is not. A real life example of yBorrowable $>$ yMin is in some cases one could borrow up to 110% of the housing value when applying for the mortgage. The only constraint that is brought by can-not-die-in-debt is $\mathfrak{h}\text{BorrowableLife}_t \leq \mathfrak{h}\text{MinLife}_t$, i.e. the accumulated over period. It is not necessarily that we need to have $y\text{Borrowable}_t \leq y\text{Min}_t$ every period, except for the last period.

$$\begin{aligned}
\mathfrak{h}\text{MinLife}_t &= \sum_{s=t+1}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)y\text{Min}_s && (32)\\
\blacktriangle\mathfrak{h}\text{MinLife}_t &= \mathfrak{h}\text{ExpLife}_t - \mathfrak{h}\text{MinLife}_t && (33)\\
&= \sum_{s=t+1}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)(y\text{Exp}_s - y\text{Min}_s) && (34)\\
\blacktriangle\mathfrak{h}\text{BorrowableLife}_t &= \mathfrak{h}\text{ExpLife}_t - \mathfrak{h}\text{BorrowableLife}_t && (35)\\
&= \sum_{s=t+1}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)(y\text{Exp}_s - y\text{Borrowable}_s) && (36)\\
\blacktriangle\mathfrak{h}\text{AccessibleLife}_t &= \mathfrak{h}\text{ExpLife}_t - \mathfrak{h}\text{AccessibleLife}_t && (37)\\
&= (\Gamma_{t+1}/R_{t+1})(y\text{Exp}_{t+1} - y\text{Accessible}_{t+1}) && (38)\\
&\quad + \sum_{s=t+2}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)(y\text{Exp}_s - y\text{Borrowable}_s) && (39)
\end{aligned}$$

And in the middle of period we can update the elements of yExpLife, yMinLife, yBorrowableLife. Then we go back to the beginning of period and define the beginning-of-period human wealth terms:

$$\begin{aligned}
\text{hExpLife}_t &= y\text{Exp}_t + \sum_{s=t+1}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)y\text{Exp}_s && (40)\\
\text{hBorrowableLife}_t &= y\text{Borrowable}_t + \sum_{s=t+1}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)y\text{Borrowable}_s && (41)\\
\text{hMinLife}_t &= y\text{Exp}_t + \sum_{s=t+1}^{T}(\prod_{\tau=t+1}^{s}\Gamma_\tau/R_\tau)y\text{Min}_s && (42)
\end{aligned}$$

Therefore when we view things using this three-sub-period angle, we see a clear recursion:

$$\begin{aligned}
\mathfrak{h}\text{ExpLife}_t &= (\Gamma_{t+1}/R_{t+1})\text{hExpLife}_{t+1} && (43)\\
\mathfrak{h}\text{MinLife}_t &= (\Gamma_{t+1}/R_{t+1})\text{hMinLife}_{t+1} && (44)\\
\mathfrak{h}\text{BorrowableLife}_t &= (\Gamma_{t+1}/R_{t+1})\text{hBorrowableLife}_{t+1} && (45)\\
\text{hExpLife}_t &= y\text{Exp}_t + \mathfrak{h}\text{ExpLife}_t && (46)\\
\text{hMinLife}_t &= y\text{Min}_t + \mathfrak{h}\text{MinLife}_t && (47)\\
\text{hBorrowableLife}_t &= y\text{Borrowable}_t + \mathfrak{h}\text{BorrowableLife}_t && (48)
\end{aligned}$$

We also have several time-varying parameters used in solving the perfect-foresight problems:

$$\begin{aligned}
\lambda\text{Max}_t &= R_{t+1}^{1/\rho-1}\beta_{t+1}^{1/\rho} && (49)\\
\text{vSum}_t &= 1 + \lambda\text{Max}_t\text{vSum}_{t+1} && (50)\\
\kappa\text{Min}_t &= (1 + \lambda\text{Max}_t/\kappa\text{Min}_{t+1})^{-1} && (51)\\
\kappa\text{Max}_t &= (1 + \wp^{1/\rho}\lambda\text{Max}_t/\kappa\text{Max}_{t+1})^{-1} && (52)
\end{aligned}$$

Having updated these, we can proceed to actually solve the problem at period t.

2 2-Period Problem

2.1 `discreteApprox!Plot`

The first step in solving any problem involving computing the expectation of a function of a continuously distibuted random variable is to discretize the continuous distribution. `discreteApprox!Plot.m` shows an example on how the discrete approximation function works intuitively, where an equiprobable discretization is plotted for the case of $\sigma = 0.1$ as shown in Figure 1.

Note that the execution of this file only requires loading three files: (a) `setup_workspace.m` defines the approximation function and location to export the figures; (b) `setup_params.m` gives the value of σ, the standard error of the mean-one log-normal distribution; and (c) `setup_shocks.m` executes the approximation function. Hence we do not load other `setup` files, which are not necessary at this moment.

2.2 `2Period`

Now we are in a position to look at Section 5.2. Here the two-period problem is solved by `2Period.m` and the solid line in Figure 2 and 3 are plotted by `2period!Plot.m`. In `2Period.m`, we first load the master `setup_everything.m`, then load `prepare.m`, which defines a two-step process to solve for the optimal consumption function at period $T-1$, and lastly run the solution process which is defined by a function called `SolveAnotherPeriod`. Details on the two-step process are as below:

- first, `functions_stable.m` defines five functions, which is to relate the consumption and value functions at the end of time t ($\mathfrak{v}(a_t)$, $\mathfrak{v}'(a_t)$, $\mathfrak{v}''(a_t)$, $\mathfrak{c}(a_t)$ and $\mathfrak{c}'(a_t)$) to the corresponding counterparts at the beginning of time $t+1$. The detailed derivations are as follows:

$$
\begin{aligned}
m_{t+1,i} &= \frac{R}{\Gamma} a_t + \theta_{t+1,i} && (53)\\
\mathfrak{v}_t(a_t) &= \beta \Gamma^{1-\rho} \left(\frac{1}{n}\right) \sum_{i=1}^{n} \mathrm{v}_{t+1}(m_{t+1,i}) && (54)\\
\mathrm{v}'_{t+1}(m_{t+1,i}) &= u\mathrm{P}\big(\mathrm{c}_{t+1}(m_{t+1,i})\big) && (55)\\
\mathfrak{v}'_t(a_t) &= \beta \Gamma^{1-\rho} \left(\frac{1}{n}\right) \sum_{i=1}^{n} \mathrm{v}'_{t+1}(m_{t+1,i}) R/\Gamma && (56)\\
&= \beta R \Gamma^{-\rho} \left(\frac{1}{n}\right) \sum_{i=1}^{n} u\mathrm{P}\big(\mathrm{c}_{t+1}(m_{t+1,i})\big) && (57)\\
\mathrm{v}''_{t+1}(m_{t+1,i}) &= u\mathrm{PP}\big(\mathrm{c}_{t+1}(m_{t+1,i})\big) \kappa_{t+1}(m_{t+1,i}) && (58)
\end{aligned}
$$

$$\mathfrak{v}_t''(a_t) = \beta R^2 \Gamma^{-\rho-1} \left(\frac{1}{n}\right) \sum_{i=1}^{n} \mathrm{v}_{t+1}''(m_{t+1,i}) \tag{59}$$

$$= \beta R^2 \Gamma^{-\rho-1} \left(\frac{1}{n}\right) \sum_{i=1}^{n} u\mathrm{PP}\big(\mathrm{c}_{t+1}(m_{t+1,i})\big)\kappa_{t+1}(m_{t+1,i}) \tag{60}$$

$$\mathfrak{c}(a_t) = n\mathrm{P}\big(\mathfrak{v}_t'(a_t)\big) \tag{61}$$

$$\mathfrak{c}'(a_t) = -1/\rho\big(\mathfrak{v}_t'(a_t)\big)^{-1/\rho-1}\mathfrak{v}_t''(a_t) \tag{62}$$

$$= \frac{\mathfrak{v}_t''(a_t)}{u\mathrm{PP}\Big(\big(\mathfrak{v}_t'(a_t)\big)^{-1/\rho}\Big)} \tag{63}$$

- second, with these end-of-period functions, we can define the optimal consumption function: (1) realize the lower bound of end-of-period asset is in $\mathfrak{m}$LowerBoundLife, (2) notice that the maximal resource available for consumption is $\blacktriangle m_t = m_t - \mathfrak{m}\mathrm{LowerBound}_t$, (3) search for the optimal consumption between a lower and an upper bound[9] by calling `FindMaximum` algorithm. After this we define the value function, which is trivial by substituting the optimal consumption into the Bellman equation (the summation of current period utility from c_t and the end-of period value from a_t).

This is the flow of thought for the two-step solution process. Each time when asked about for the optimal consumption at an arbitrary level of cash-on-hand, the computer must start the `FindMaximum` process, which is time consuming, albeit mostly accurately. Hence there is a huge amount of inefficiencies here. Moreover when consumption functions are defined this way, it is almost impossible to solve for the optimal consumption at period $T-2$, not even to mention periods further forward.

Of course in between the above two steps, we need to update various lists, which is done in a `SolveAnotherPeriod` function[10], and which in turn calls the `AddNewPeriodToParamLifeDates` function and also increases the iterator `PeriodsSolved` by one, i.e. updates the solution one period forward.

So here is the structure of the `prepare.m`: we load (1) `functions_stable.m`, (2) define the consumption and value function, which details the procedure to solve the problem via the `FindMaximum` algorithm, (3) define a `SolveAnotherPeriod` function.

In the end, `2Period!Plot.m` loads `2Period.m` and plots the value and consumption for $T-1$, as reflected in the solid line in Figure 2 and 3.

2.3 `2PeriodInt`

We proceed to Section 5.3, where the two-period problem is solved by `2PeriodInt.m` and the dashed line in Figure 2 and 3 are plotted by `2periodInt!Plot.m`. The `Int` is used as

[9] Here we use a very arbitrary search range. Actually we could use the consumption function for a pessimist and for an optimist, and hence have a narrower and more efficient range than $[0, \blacktriangle m_t]$.

[10] In essence `SolveAnotherPeriod` here does not materially solve any optimization problem; it simply updates the problem backward another period.

a postfix because here the consumption and value functions are `InterpolatingFunction` objects in the language of `Mathematica`: they are derived by connecting a small number of grid points, as compared to the one in `2Period.m` (which needs to call brute-force `FindMaximum` algorithm every time). The program structure resembles that of `2Period.m`, with the following two differences:

- First in `SolveAnotherPeriod` function, we solve for the optimal consumptions at a number of given grid points of cash-on-hand, by involving the `FindMaximum` algorithm, and then do an interpolation along the grid points;
- Second in `functions_Interpolate.m`, when asked about the optimal consumption at an arbitrary point, the computer will not start the time-consuming `FindMaximum` process again; instead, it will look for the answer based on the interpolation function which is generated in `SolveAnotherPeriod` and which is much faster, although less accurate.

Here is the point: we essentially separate the previous definition of consumption function into two steps: first do the `FindMaximum` for a number of grid points, and second for any other points we will rely on the interpolation-based functions. This is an approximation, and it saves tremendous amount of computing time while reduces the accuracy. As we see, this raw interpolation is pretty inaccurate, and the ensuing programs are designed to improve the accuracy of the interpolation, while not sacrificing the huge efficiency gain associated.

2.4 2PeriodIntExp

We look at Section 5.4, where the two-period problem is solved by `2PeriodIntExp.m` and Figure 4, 5 and 6 are plotted by `2periodIntExp!Plot.m`. The `Exp`[11] is used as a postfix because here we solve for optimal consumption by interpolating the $\mathfrak{v}_t(a_t)$ first. Several changes are listed below:

- The `2PeriodIntExp.m` adds the initialization of a new list `vFuncLife`, since we will append our interpolation function to it.
- Within the `SolveAnotherPeriod` function, we first do an interpolation, append it to `vFuncLife`, and replace the true end-of-period function with the interpolated one `vHat` when we do `FindMaximum`.
- How `vHat` is derived from `vFuncLife`, is defined in `functions_IntExpGothicV.m`.

2.5 2PeriodIntExpFOC

In Section 5.5, the two-period problem is solved by `2PeriodIntExpFOC.m` and Figure 7 and 8 are plotted by `2periodIntExpFOC!Plot.m`. The `FOC` is used as a postfix because

[11] I think the `Exp` is somewhat confusing. Is is better if we use `EndV` (end of period value function)?

here we solve for optimal consumption by utilizing the first order conditions, where we approximate the marginal value function. The changes are:

- The `2PeriodIntExpFOC.m` adds the initialization of a new list $\mathfrak{v}$aFuncLife, from which we can derive $\mathfrak{v}$aHat as the interpolated marginal end-of-period value function.
- Within the `SolveAnotherPeriod` function, we first do an interpolation to be appended to $\mathfrak{v}$`aFuncLife`, and use $\mathfrak{v}$aHat in the root-finding process. Note that the algorithm is to use `FindRoot` to find the solution to the first-order-condition, as opposed to use the `FindMaximum` to find the arg-max to the optimization problem. This saves much computational time.

2.6 2PeriodIntExpFOCInv

In section 5.6 we approximate the marginal value of asset at the end of period by linearly interpolating between the asset grid and $\mathfrak{c}_t(a_t) = [\mathfrak{v}'_t(a_t)]^{-1/\rho}$ (i.e. an inverted version of $\mathfrak{v}'_t(a_t)$). In Section 5.7 the self-imposed liquidity constraint implies that we need to augment the vectors by including the lower bound of asset and zero consumption. In Section 5.9 we realize for a given end-of-period asset, the optimal consumption is given by $\mathrm{c}_t = \mathfrak{c}_t(a_t)$, and we can recover the cash-on-hand by $m_t = \mathfrak{c}_t + a_t$. The two-period problem is solved by `2PeriodIntExpFOCInv.m` and Figure 9 and 10 are plotted by `2periodIntExpFOCInv!Plot.m`. The `Inv` is used as a postfix because here we solve for optimal consumption by utilizing the endogenous gridpoint method, i.e. instead of calling the `FindMaximun` or `FindRoot` function, we invert the marginal value of asset at the end of period, to arrive at the optimal consumption. The changes are

- The `2PeriodIntExpFocInv.m` adds the initialization of a new list $\mathfrak{c}$FuncLife, since we will append our interpolated consumption as a function of the end-of-period asset to it.
- Within the `SolveAnotherPeriod` function, the only change is to (1) start with a list of end-of-period grid point α`Vec`, (2) generate the corresponding $\mathfrak{v}$`aVect`, (3) recover the optimal consumption vector by inverting $\mathfrak{v}$`aVect`, (4) augment the asset and consumption vector by including the lowest possible asset value and zero consumption, (5) recover the cash-on-hand grid by the simple accounting rule, and (6) construct two functions $\mathrm{c}_t(m_t)$ (the consumption function) and $\mathfrak{c}_t(a_t)$ (the consumed function) by linear interpolation. Note this endogenous grid point method does not involve any root-finding process and thus is more efficient.

2.7 2PeriodIntExpFOCInvEEE

We come to Section 5.8 where we construct an asset grid that is more densely spaced at lower values, as opposed to an even distribution previously. Here we use the pragmatic choice of a *multi-exponential* growth rate. The two-period problem

is solved by `2PeriodIntExpFOCInvEEE.m` and Figure 11 and 12 are plotted by `2periodIntExpFOCInvEEE!Plot.m`. The `EEE` is used as a postfix because here we construct the asset and cash grid by a triple-exponential growth pattern; and the details are shown in `setup_grids_eee.m`, which is the only change we make over the previous version.

2.8 `ExtrapProblem!Plot`

There is also a `ExtrapProblem!Plot.m`, which is to draw Figure 13. This is done by loading `2periodIntExpFOCInv.m`[12] and `DefinecTm1Raw.m`. The second file in turn defines several functions, among which we will use is $\mathrm{c}_{T-1}^{\mathrm{Raw}}(m_t)$ (through calling `FindRoot` function to solve the first-order condition, and specifying appropriate upper and lower bounds for solution searching). Several other associated functions are defined, but not used currently, like $\varrho_{T-1}^{\mathrm{Raw}}(\mu_t)$ and $\chi_{T-1}^{\mathrm{Raw}}(\mu_t)$.

2.9 `2PeriodIntExpFOCInvPesReaOpt`

We come to Section 5.10 and 5.11 where we approximate the precautionary saving instead of optimal consumption, and apply a similar trick to the value function. The two-period problem is solved by `2PeriodIntExpFOCInvPesReaOpt.m` and Figure 14 and 15 are plotted by `2periodIntExpFOCInvPesReaOpt!Plot.m`. The `PesReaOpt` is used as a postfix because here when we construct the precautionary saving, we rely on the solution to the problem of a pessimist, a realist and an optimist.

In the `setup` part, we improve the asset grid a little bit further using a multi-exponential growth pattern in `setup_grids_expMult.m`, and load `setup_lastperiod_PesReaOpt.m` which initializes various lists for the method of moderation step. There is a setup of four parameters like μSmallGapLeft, which are used as a small trick, to expediate a linear extrapolation outside the interpolation grid range[13].

The big change is in the definition of `SolveAnotherPeriod` function, which replies on two files. First we look at the `AddNewPeriodToSolvedLifeDates.m`:

- It loads the lower bound of the cash-on-hand and end-of-period asset vectors;
- It defines the $\mathfrak{a}$Vect through augmenting the $\mathfrak{a}$Vec with a point very close to its lower bound, which makes sure that the asset grid is very dense near the lower bound;
- If there is an artificial constraint, then it adds the point where the constraint will be binding to the $\mathfrak{a}$Vect.

[12] The most natural way is to load `2periodIntExpFOCInvEEE.m`, which will show that the extrapolation problem becomes serious when m exceeds to 100. While here we load `2periodIntExpFOCInv.m`, and show that the extrapolation problem becomes serious as long as m is larger than 10. This conveys the same message more vividly, yet under a parsimonious choice.

[13] Basically we would like to have linear (as opposed to the default spline) extrapolation below the smallest and above the highest grid points; and this is done by augment the pair of grid points by a small step below the smallest and above the highest points.

- It calculates the corresponding vectors of $\mathfrak{v}$, $\mathfrak{v}^a$ and $\mathfrak{v}^{aa}$;
- It calculates the following:

$$\begin{aligned}
\mathrm{c} &= \mathfrak{c}(a) && (64)\\
&= n\mathrm{P}(\mathfrak{v}^a) && (65)\\
\mathfrak{c}^a &= \frac{\partial(\mathfrak{v}^a)^{-1/\rho}}{\partial a} && (66)\\
&= -\rho^{-1}(\mathfrak{v}^a)^{-1/\rho-1}\mathfrak{v}^{aa} && (67)\\
&= \frac{\mathfrak{v}^{aa}}{u\mathrm{PP}\big(n\mathrm{P}(\mathfrak{v}^a)\big)} && (68)\\
&= \frac{\mathfrak{v}^{aa}}{u\mathrm{PP}(\mathrm{c})} && (69)\\
\mathrm{m} &= \mathrm{c} + \mathfrak{a} && (70)\\
\mathrm{m}^a &= \mathfrak{c}^a + 1 && (71)\\
\mathrm{m}^a &= \kappa_m \mathrm{m}^a + 1 && (72)\\
\kappa_m &= \frac{\mathrm{m}^a - 1}{\mathrm{m}^a} && (73)\\
&= \frac{\mathfrak{c}^a}{\mathfrak{c}^a + 1} && (74)
\end{aligned}$$

- It defines $\blacktriangle m$, μ and $\underline{\mathrm{c}}$;
- It generates the value vector and augments various vectors by including their lower bound value;
- It does linear interpolation to generate the value function, the consumption function and the consumed function.

Second we look at `AddNewPeriodToSolvedLifeDatesPesReaOpt.m`, which is as follows:

- It loads $\blacktriangle \mathrm{h}_t$, $\underline{\kappa}_t$;
- It defines the consumption vector for a realist and for an optimist;
- It defines ϱ, ϱ^μ, χ and χ^μ:

$$\begin{aligned}
\varrho_t(\mu_t) &= \varrho_t(\blacktriangle \mathfrak{m}_t) && (75)\\
&= \frac{\underline{c}_t(\blacktriangle \mathfrak{m}_t + \blacktriangle \mathrm{h}_t) - c_t(\blacktriangle \mathfrak{m}_t + \blacktriangle \mathrm{h}_t)}{\underline{\kappa}_t \blacktriangle \mathrm{h}_t} && (76)\\
\varrho^\mu_t(\mu_t) &= \frac{(\underline{\kappa}_t - \kappa_t)\blacktriangle \mathfrak{m}_t}{\underline{\kappa}_t \blacktriangle \mathrm{h}_t} && (77)\\
\chi_t(\mu_t) &= \log\Big(\frac{1 - \varrho_t(\mu_t)}{\varrho_t(\mu_t)}\Big) && (78)
\end{aligned}$$

$$\chi_t^{\mu}(\mu_t) = -\frac{\varrho_t^{\mu}(\mu_t)}{\varrho_t(\mu_t)\big(1-\varrho_t(\mu_t)\big)} \tag{79}$$

- It interpolates to generate the $\varrho(\mu)$ and $\chi(\mu)$ functions;
- It define vSum_t, $\bar{\Lambda}$ and $\bar{\Lambda}^m$, Λ and Λ^m:

$$\begin{aligned}
\bar{\text{v}}_t(m) &= u\big(\bar{c}_t(m)\big)\text{vSum}_t && (80)\\
\bar{\Lambda}_t(m) &\equiv \big((1-\rho)\bar{\text{v}}_t(m)\big)^{1/(1-\rho)} && (81)\\
&= \bar{c}_t(m)(\text{vSum}_t^T)^{1/(1-\rho)} && (82)\\
\bar{\Lambda}_t^m(m) &= \kappa\text{Min}_t(\text{vSum}_t^T)^{1/(1-\rho)} && (83)\\
\Lambda_t(m) &\equiv \big((1-\rho)\text{v}_t(m)\big)^{1/(1-\rho)} && (84)\\
\Lambda_t^m(m) &= \big((1-\rho)\text{v}_t(m)\big)^{1/(1-\rho)-1}\text{v}_t^m(m) && (85)\\
&= \big((1-\rho)\text{v}_t(m)\big)^{1/(1-\rho)-1}u\text{P}\big(c_t(m)\big) && (86)\\
& && (87)
\end{aligned}$$

- It defines $\Omega_t(\mu)$, $\Omega_t^{\mu}(\mu)$, $\text{X}_t(\mu)$ and $\text{X}_t^{\mu}(\mu)$:

$$\begin{aligned}
\Omega_t(\mu) &= \frac{\bar{\Lambda}_t(e^{\mu})-\Lambda_t(e^{\mu})}{\underline{\kappa}_t\blacktriangle\text{h}_t(\text{vSum}_t^T)^{1/(1-\rho)}} && (88)\\
\Omega_t^{\mu}(\mu) &= e^{\mu}\frac{\bar{\Lambda}_t^m(m)-\Lambda_t^m(m)}{\underline{\kappa}_t\blacktriangle\text{h}_t(\text{vSum}_t)^{1/(1-\rho)}} && (89)\\
\text{X}_t(\mu) &= \log\Big(\frac{1-\Omega_t(\mu)}{\Omega_t(\mu)}\Big) && (90)\\
\text{X}_t^{\mu}(\mu) &= -\frac{\Omega_t^{\mu}(\mu)}{\Omega_t(\mu)\big(1-\varrho_t(\mu)\big)} && (91)
\end{aligned}$$

- It interpolates to create the $\Lambda(m)$, $\Omega_t(\mu)$, and $\text{X}_t(\mu)$ functions.

Third `functionsIntExpFOCInvPesReaOpt.m`, which is an overhaul of the definition of consumption function in `functions_Interpolate.m`, generates various functions:

- given the interpolated function $\hat{\chi}_t(\mu_t)$, we need to derive the consumption function and its first derivative[14]:

$$\begin{aligned}
\hat{\chi}_t^{\mu}(\mu) &= \frac{\partial\hat{\chi}_t(\mu)}{\partial\mu} && (92)\\
\hat{\varrho}_t(\mu) &= \frac{1}{1+e^{\hat{\chi}_t(\mu)}} && (93)\\
\hat{\varrho}_t^{\mu}(\mu) &= \frac{\partial\hat{\varrho}_t(\mu)}{\partial\mu} && (94)
\end{aligned}$$

[14]For various transformations between the consumption and value functions, and χ and X, there is a `Derivations.nb` to serve as a symbolic check-up.

$$
\begin{aligned}
&= -\frac{e^{\hat{\chi}_t(\mu)}\hat{\chi}_t^{\mu}(\mu)}{\big(1+e^{\hat{\chi}_t(\mu)}\big)^2} && (95)\\
&= \big(\hat{\varrho}_t(\mu)-1\big)\hat{\varrho}_t(\mu)\hat{\chi}_t^{\mu}(\mu_t) && (96)\\
\blacktriangle m_t &= m_t + \underline{\mathfrak{h}}_t && (97)\\
\mu_t &= \log(\blacktriangle m_t) && (98)\\
\hat{c}_t(m_t) &= \underline{c}_t(m_t) - \hat{\varrho}_t(\mu_t)\underline{\kappa}_t\blacktriangle\mathfrak{h}_t && (99)\\
\hat{\kappa}_t(m_t) &= \frac{\partial\hat{c}_t(m_t)}{\partial m_t} && (100)\\
&= \underline{\kappa}_t - \frac{\hat{\varrho}_t^{\mu}(\mu_t)}{\blacktriangle m_t}\underline{\kappa}_t\blacktriangle\mathfrak{h}_t && (101)\\
& && (102)
\end{aligned}
$$

- given the interpolated function $\hat{\mathrm{X}}_t(\mu_t)$, we derive the value function and its first derivatives[15]:

$$
\begin{aligned}
\hat{\mathrm{X}}_t^{\mu}(\mu) &= \frac{\partial\hat{\mathrm{X}}_t(\mu)}{\partial\mu} && (103)\\
\hat{\Omega}_t(\mu) &= \frac{1}{1+e^{\hat{\mathrm{X}}_t(\mu)}} && (104)\\
\hat{\Omega}_t^{\mu}(\mu) &= \frac{\partial\hat{\Omega}_t(\mu)}{\partial\mu} && (105)\\
&= -\frac{e^{\hat{\mathrm{X}}_t(\mu)}\hat{\mathrm{X}}_t^{\mu}(\mu)}{\big(1+e^{\hat{\mathrm{X}}_t(\mu)}\big)^2} && (106)\\
&= \big(\hat{\Omega}_t(\mu)-1\big)\hat{\Omega}_t(\mu)\hat{\mathrm{X}}_t^{\mu}(\mu) && (107)\\
\blacktriangle m_t &= m_t + \underline{\mathfrak{h}}_t && (108)\\
\mu_t &= \log(\blacktriangle m_t) && (109)\\
\hat{\Lambda}_t(m_t) &= \bar{\Lambda}_t(m_t) - \hat{\Omega}_t(\mu_t)\underline{\kappa}_t\blacktriangle\mathfrak{h}_t(\mathbb{C}_t^T)^{1/(1-\rho)} && (110)\\
\hat{\Lambda}_t^m(m_t) &= \underline{\kappa}_t(\mathbb{C}_t^T)^{1/(1-\rho)} - \frac{\hat{\Omega}_t^{\mu}(\mu_t)}{\blacktriangle m_t}\underline{\kappa}_t\blacktriangle\mathfrak{h}_t(\mathbb{C}_t^T)^{1/(1-\rho)} && (111)\\
\hat{\mathrm{v}}_t(m_t) &= u\big(\hat{\Lambda}_t(m_t)\big) && (112)\\
\hat{\mathrm{v}}_t^m(m_t) &= u\mathrm{P}\big(\hat{c}_t(m_t)\big) && (113)
\end{aligned}
$$

2.10 `ExtrapProblemSolved!Plot`

When we finish with the method of moderation, we would like to see whether the extrapolation problem, which is the motive behind this transformation, is solved or not. `ExtrapProblemSolved!Plot.m` is designed to draw Figure 14, as a comparison to Figure

[15] In deriving the marginal value function, we use the Envelope Theorem directly. A more intuitive but cumbersome way will be $\hat{\mathrm{v}}_t^m(m_t) = u\mathrm{P}\big(\hat{\Lambda}_t(m_t)\big)\hat{\Lambda}_t^m(m_t)$.

13. This is done by loading `2periodIntExpFOCInvPesReaOpt.m` and `DefinecTm1Raw.m`, and by plotting their difference with the perfect-foresight solution.

2.11 2PeriodIntExpFOCInvPesReaOptTighterUpBd

In the above method of moderation, we treat the optimist's consumption rule as the upper bound. But in fact, there is another upper bound which is determined by the existence of natural borrowing constraint. This is explicitly dealt with in `2PeriodIntExpFOCInvPesReaOptTighterUpBd.m`.

First we have a `setup_lastperiod_PesReaOptTighterUpBd.m`, which is to initialize various lists, which we will append to during the solution process.

Second we have a `setup_PerfectForesightSolutionTighterUpBd.m`, which is to define the consumption and value functions for a consumer who faces the same income environment as the perfect-foresight optimist, but chooses to spend according to the consumption rule in the tighter upper bound. While the consumption function is very straightforward, we have to define the end-of-period value function in a similarly recursive way, and derive the start-of-period value function through the interpolation. The big challenge is that, although we do have the analytical consumption function, we do not have an analytical solution for the value function.

Third we have a `AddNewPeriodToParamLifeDatesTighterUpBd.m`, which updates several lists related to the `TighterUpBd` process.

Fourth we have a `AddNewPeriodToSolvedLifePesReaOptTighterUpBd.m`, which (1) renames the those previous defined functions using a postfix `Hi`, (2) separates the grids around the cusp point to three parts, (3) defines the corresponding functions using a postfix Lo and Md, and (4) approximates the value function for those behaving according to the Tighter Upper Bound rules.

2.12 2PeriodIntExpFOCInvPesReaOptTighterUpBdCon

All the above are assuming that the only constraint is the natural borrowing constraint. Now if we impose an artificial constraint, which means the borrowable income next period is smaller than the minimum income like the unemployment benefits. Then we need to consider the constrained case. A common case is any borrowing from the future is not allowed at all, and hence the end-of-period asset can not be smaller than zero, even if the worst case scenario is agent will still have employment benefit payments from now on.

This is where the distinction between ℌAccessibleLife and ℌBorrowableLife makes sense: the former defines the maximal accessible amount to an unconstrained consumer this period (which is the lower bound for his/her end-of-period asset) and the later defines the maximal borrowable amount to a constrained consumer (if cash-on-hand is below which, the liquidity constraint will become binding).

Compared to the unconstrained case, several changes are:

- In the `2PeriodIntExpFOCInvPesReaOptTighterUpBdCon.m`, we redefine yBorrowable to be zero[16].
- We also define a `setup_lastperiod_PesReaOptTighterUpBdCon.m`, which initializes various lists that are related to capture the star point, at which the artificial liquidity constraint becomes exactly binding.
- We load an additional `AddNewPeriodToSolvedLifeDatesPesReaOptTighterUpBdCon.m` which is to compute the consumption, asset, cash, value etc. at the point where the artificial liquidity constraint becomes exactly binding.
- We also revise the definition of consumption functions etc in `functions_PesReaOptTighterUpBdC` and the central revision is on `cBeforeT` which is piecewise below and above the star point.

So what does an artificial liquidity constraint bring about? Essentially it only adds one additional step: after deriving the consumption function for a period-t-unconstrained consumer who is the same as the constrainted consumer except that s/he can borrow against next period's minimum income, then the consumption function for the period-t-constrained consumer will be a piecewise one, below which the constraint becomes binding; and the particularly nice thing is we can pinpoint the exact point where this happens.

3 Other Files

3.1 `multiperiod`

This file is the same as `2periodIntExpFOCInvPesReaOptTighterUpBd.m`, except that we load `setup_params_multiperiod.m`, which is to redefine the variance of transitory shock to be a lower and more realistic value. And `multiperiod!Plot.m` executes `SolveAnotherPeriod` many times, and plots the convergence of consumption series.

3.2 `multiperiodCon`

This file is the multiperiod version of `2periodIntExpFOCInvPesReaOptTighterUpBdCon.m`, except that the `setup_params_multiperiod.m`. And `multiperiodCon!Plot.m` plots the convergence of consumption series as in Figure 17.

[16]This is an extreme. With yMin $= 0.05$ as we assume, any yBorrowable that is less than 0.05 constitutes an artificial liquidity constraint, and we can deal with that, by just change this parameter value.

3.3 `multicontrol`

This is to solve for an optimal choice of consumption and risky asset allocation. The current version utilizes the above 2-period techniques up to `2periodIntExpFOCInv` (i.e. no method of moderation is used).

3.4 `Notation`

This explains various naming conventions, like the human wealth, perfect-foresight marker, MPC, etc.