Numerical Methods - Final Project

Jiacheng Li

April, 2022

Contents

1 Lucas and Stokey (1983)

1.1 Environment

Let $t \geq 0$. Let (Π, π_0) be a finite space Markov chain with initial distribution at time 0, $\pi_0\left(s_0\right)$, and state space $\mathcal{S} \equiv \{1,2,...,S\} \ni s$. A history is denoted by s^t and the joint density over s^t induced by s_0, (Π, π_0) .

- Government
  The government issues a one-period Arrow state-contingent debt conditional on history s^t at time t that pays 1 unit at $t+1$, with market-determined state-contingent price $p_{t+1}\left(s_{t+1}|s^t\right)$. It has a given initial debt $b_0\left(s_0\right)$.
  It has an exogenous stream of state-contingent public spending $\left\{\left\{g_t\left(s^t\right)\right\}_{\forall s^t}\right\}_{\forall t\geq 0}$.
  It imposes a flat-rate tax $\tau_t\left(s^t\right)$ on labor income conditional on s^t at time t.
  Thus, the government has budget constraint:

$$g_t\left(s_t\right) = \tau_t^n\left(s^t\right) n_t\left(s^t\right) + \sum_{s_{t+1}} p_{t+1}\left(s_{t+1} \mid s^t\right) b_{t+1}\left(s_{t+1} \mid s^t\right) - b_t\left(s_t \mid s^{t-1}\right)$$

- Technology

  The economy features production technology $y_t\left(s^t\right)=n_t\left(s^t\right)$, so the wage rate is simply 1.

  There is feasibility constraint/market clearing:

$$c_t\left(s^t\right)+g_t\left(s^t\right)=n_t\left(s^t\right), \forall t$$

- Preferences

  Representative household would like to solve

$$\max\ \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u\left(c_t, l_t\right)=\sum_{t=0}^{\infty} \sum_{s^t} \beta^t \pi_t\left(s^t\right) u\left(c_t\left(s^t\right), l_t\left(s^t\right)\right)$$

  subject to

$$c_t\left(s^t\right)+\sum_{s_{t+1}} p_t\left(s_{t+1} \mid s^t\right) b_{t+1}\left(s_{t+1} \mid s^t\right)=\left[1-\tau_t\left(s^t\right)\right] n_t\left(s^t\right)+b_t\left(s_t \mid s^{t-1}\right) \quad \forall t \geq 0$$

$$n_t\left(s^t\right)+l_t\left(s^t\right)=1 \quad \forall t \geq 0$$

The price system is equivalent to a Arrow-Debreu price system under time-0 trading with prices of Arrow-Debreu securities given by

$$q_{t+1}^0\left(s^{t+1}\right)=p_{t+1}\left(s_{t+1} \mid s^t\right) q_t^0\left(s^t\right). \tag{1}$$

And we can choose numeraire $q_0^0\left(s^0\right)=1$.

1.2 Competitive Equilibrium

Definition 1.1. A competitive equilibrium with taxes is a feasible allocation $\left\{c_t\left(s^t\right), n_t\left(s^t\right)\right\}_{t=0}^{\infty}$, a price system $\left\{p_{t+1}\left(s_{t+1} \mid s^t\right)\right\}_{t=0}^{\infty}$, and a government policy $\left\{g_t\left(s^t\right), \tau_t\left(s^t\right), b_{t+1}\left(s^{t+1}\right)\right\}_{t=0}^{\infty}$ such that:

1) Given the price system and government policy, the allocation solves the households's problem.

2) Government's budget constraint is satisfied at all times.

The first order condition of household's problem implies

$$\left[l_t\left(s^t\right)\right] \qquad 1-\tau_t\left(s^t\right)=\frac{u_l\left(s^t\right)}{u_c\left(s^t\right)}, \forall t, s^t \tag{2}$$

$$\left[b_t\left(s_{t+1} \mid s^t\right)\right] \qquad p_{t+1}\left(s_{t+1} \mid s^t\right)=\beta \pi\left(s_{t+1} \mid s^t\right)\left(\frac{u_c\left(s^{t+1}\right)}{u_c\left(s^t\right)}\right), \forall t, s^t \tag{3}$$

Under the Arrow-Debreu trading structure, iterative substitution of household budget constraint yields the present (time-0) value budget constraint:

$$\sum_{t=0}^{\infty} \sum_{s^t} q_t^0\left(s^t\right) c_t\left(s^t\right)=\sum_{t=0}^{\infty} \sum_{s^t} q_t^0\left(s^t\right)\left[1-\tau_t\left(s^t\right)\right] n_t\left(s^t\right)+b_0$$

And (1), (3) yield

$$q_t^0\left(s^t\right)=\beta^t \pi_t\left(s^t\right) \frac{u_c\left(s^t\right)}{u_c\left(s^0\right)}$$

Substitution under FOC (2) and (3) to remove taxes and prices allows us to derive the following implementability condition:

$$\sum_{t=0}^{\infty} \sum_{s^t} \beta^t \pi_t\left(s^t\right)\left[u_c\left(s^t\right) c_t\left(s^t\right)-u_{\ell}\left(s^t\right) n_t\left(s^t\right)\right]-u_c\left(s^0\right) b_0=0 \tag{4}$$

1.3 Ramsey Problem

The government would like to choose a feasible allocation$\{c_t(s^t), n_t(s^t)\}_{t=0}^{\infty}$ that solves the household's problem, i.e.,

$$\max \sum_{t=0}^{\infty}\sum_{s^t}\beta^t\pi_t(s^t)\,u(c_t(s^t), 1-n_t(s^t))$$

subject to (4).

This yields the Lagrangian

$$\begin{aligned}\mathcal{L} = \sum_{t=0}^{\infty}\sum_{s^t}\beta^t\pi_t(s^t)&\left\{u(c_t(s^t), 1-n_t(s^t)) + \theta_t(s^t)\left[n_t(s^t) - c_t(s^t) - g_t(s^t)\right]\right.\\ &\left.+\lambda\left[u_c(s^t)\,c_t(s^t) - u_\ell(s^t)\,n_t(s^t)\right]\right\} - \lambda u_c\left(s^0\right)b_0\end{aligned}$$

where λ is the multiplier on the single implementability constrain, and $\{\theta_t(s^t)\}_{\forall t,s^t}$ is the sequence of multipliers on all feasibility constraints.

And this leads to the following sets of first-order conditions:

$$\begin{aligned}&[c_t(s^t)] &&(1+\lambda)u_c(s^t) + \lambda\left[u_{cc}(s^t)\,c_t(s^t) - u_{lc}(s^t)\,n_t(s^t)\right] - \theta_t(s^t) = 0, \quad t\geq 1\\ &[n_t(s^t)] &&-(1+\lambda)u_l(s^t) - \lambda\left[u_{cl}(s^t)\,c_t(s^t) - u_{ll}(s^t)\,n_t(s^t)\right] + \theta_t(s^t) = 0, \quad t\geq 1\end{aligned}$$

And,

$$\begin{aligned}&\left[c_0\left(s^0, b_0\right)\right] &&(1+\lambda)u_c\left(s^0, b_0\right) + \lambda\left[u_{cc}\left(s^0, b_0\right)c_0\left(s^0, b_0\right) - u_{\ell c}\left(s^0, b_0\right)n_0\left(s^0, b_0\right)\right]\\ & &&-\theta_0\left(s^0, b_0\right) - \lambda u_{cc}\left(s^0, b_0\right)b_0 = 0\\ &\left[n_0\left(s^0, b_0\right)\right] &&-(1+\lambda)u_\ell\left(s^0, b_0\right) - \lambda\left[u_{c\ell}\left(s^0, b_0\right)c_0\left(s^0, b_0\right) - u_{\ell\ell}\left(s^0, b_0\right)n_0\left(s^0, b_0\right)\right]\\ & &&+\theta_0\left(s^0, b_0\right) + \lambda u_{cc}\left(s^0, b_0\right)b_0 = 0\end{aligned}$$

Notice that the first set of conditions can yield

$$(1+\lambda)u_c(s^t) + \lambda\left[u_{cc}(s^t)\,c_t(s^t) - u_{lc}(s^t)\,n_t(s^t)\right] - (1+\lambda)u_l(s^t) - \lambda\left[u_{cl}(s^t)\,c_t(s^t) - u_{ll}(s^t)\,n_t(s^t)\right] = 0$$

where $u_{cc}(s^t), u_{lc}(s^t)$ and $u_{ll}(s^t)$ are nothing but functions of $(c_t(s^t), n_t(s^t)) = (c_t(s^t), 1-g_t(s^t)-c_t(s^t))$ pair. A consequence of the above equation is that the allocation of consumption and hours is an implicit function of $g_t(s^t)$. That is, for any two different paths of history, as long as $g_t(s^t) = g_t(\tilde{s}^t)$, the optimal allocation should be the same.

One more observation is that, the above first order conditions yield different results for $t=0$ and $t\geq 1$. When $t=0$, the allocation c and n also depend on the government's initial debt b_0. This is connected to the time inconsistency of the Ramsey problem.

1.4 Recursive Formulation

From now on, we impose the condition that government spending $g(s^t)$ is a time-invariant function of the current state s_t.

Recall that the Ramsey planner has to satisfy budget constraints facing the household:

$$c_t(s^t) + \sum_{s_{t+1}} p_t(s_{t+1}\mid s^t)\,b_{t+1}(s_{t+1}\mid s^t) = \left[1-\tau_t(s^t)\right]n_t(s^t) + b_t\left(s_t\mid s^{t-1}\right) \tag{5}$$

and the implementability constraints

$$1-\tau_t(s^t) = \frac{u_l(s^t)}{u_c(s^t)}$$

$$p_{t+1}(s_{t+1}\mid s^t) = \beta\pi(s_{t+1}\mid s^t)\left(\frac{u_c(s^{t+1})}{u_c(s^t)}\right)$$

as well as the feasibility constraint

$$n_t\left(s^t\right)-c_t\left(s^t\right)-g_t\left(s^t\right)=0$$

Substitution of the above three equations into (5) and multiplying both sides by $u_c\left(s^{t+1}\right)$ yields

$$\begin{aligned}u_c\left(s_t\right)\left[n_t\left(s_t\right)-g_t\left(s_t\right)\right]+\beta\textstyle\sum_{s_{t+1}}\pi\left(s_{t+1}\mid s_t\right)u_c\left(s_{t+1}\right)b_{t+1}\left(s_{t+1}\mid s_t\right)&=\\ u_l\left(s_t\right)n_t\left(s_t\right)+u_c\left(s_t\right)b_t\left(s_t\mid s_{t-1}\right)&\end{aligned}$$

where the Markov structure is imposed.

We can define the state variable (s_t,x_t) where $x_t\equiv u_c\left(s_t\right)b_t\left(s_t\mid s_{t-1}\right)$, which represents the marginal utility scaled government debt, is the additional state. This gives a complete picture of the Ramsey planner's program at all $t\geq 1$. The recursive representation of the above condition then is

$$x=u_c\left(n-g\left(s\right)\right)-u_l n+\beta\sum_{s'\in\mathcal{S}}\pi\left(s'\mid s\right)x'\left(s'\right)$$

Note that defining the state variable implicitly requires that the planner who inherits state (s,x) commits to attain $x_t=u_c\left(s_t\right)b_t\left(s_t\mid s_{t-1}\right)$ by manipulating the current period's consumption, while choosing the future quantities x' that the future planner has to attain.

1.4.1 Subproblem 1: The Continuation Ramsey Problem

Therefore, we can define the Bellman equation for time $t\geq 1$ Ramsey planner

$$V\left(x,s\right)=\max_{n,\{x'(s')\}_{\forall s'\in\mathcal{S}}}u\left(n-g(s),1-n\right)+\beta\sum_{s'\in\mathcal{S}}\pi\left(s'\mid s\right)V\left(x',s'\right)$$

subject to

$$x=u_c\left(n-g\left(s\right),1-n\right)-u_l n+\beta\sum_{s'\in\mathcal{S}}\pi\left(s'\mid s\right)x'\left(s'\right)$$

This recursive implementability constraint defines a feasible set in which $\left(n,\{x'\left(s'\right)\}_{\forall s'\in\mathcal{S}}\right)$ is chosen.

1.4.2 Subproblem 2: The Initial Ramsey Problem

And the Bellman equation for the initial Ramsey planner is

$$W\left(b_0,s_0\right)=\max_{n_0,\{x'(s_1)\}_{\forall s_1\in\mathcal{S}}}u\left(n_0-g_0,1-n_0\right)+\beta\sum_{s_1\in\mathcal{S}}\pi\left(s_1|s_0\right)V\left(x'\left(s_1\right),s_1\right)$$

subject to

$$u_{c,0}b_0=u_{c,0}\left(n_0-g_0,1-n_0\right)-u_{l,0}n_0+\beta\sum_{s_1\in\mathcal{S}}\pi\left(s_1|s_0\right)x'\left(s_1\right)$$

1.4.3 Time Inconsistency

The fact that the time-t, history s^t continuation problem of a time-0, s_0 initial Ramsey planner is not the same as the problem of an initial Ramsey planner with initial values at time-t, history s^t manifests the time inconsistency of the Ramsey problem.

At every $t\geq 1$, given the inherited debt $b_t\left(s_t|s_{t-1}\right)$, the Ramsey planner would like to manipulate $u_c\left(s_t\right)$ as well by choosing $n_t\left(s_t\right)$. But the way the continuation problem is specified above restricts the Ramsey planner to honor the preceding Ramsey planner's choice $x_t\equiv u_c\left(s_t\right)b_t\left(s_t\mid s_{t-1}\right)$, so that the recurisve formulation of the problem is equivalent to sequential Ramsey problem at time 0.

We can indeed verify this by taking the envelop conditions on the value functions and connect them through the Lagrangian multiplier on the implementability constraints.

The recursive formulation allows us to solve the problem using value function iterations.

1.5 Numerical Implementation

The two Ramsey problem subject to constraints have defined a well-posed recursive problem to be solved with dynamic programming. In each period, the Ramsey planner chooses a set of variables

$$\left(n,\left\{x'\left(s'\right)\right\}_{\forall s'\in\mathcal{S}}\right)$$

of length $1+S$ to maximize its value function. However, one potential problem associated with implementation is that there is a single linear implementability constraint that removes one degree of freedom in the social planner's choice. To address this problem, we define the control variables to be $\left(n,\left\{x'\left(s'\right)\right\}_{s'=s_1,\ldots,s_{S-1}}\right)$ and $x'\left(s_S\right)$ is computed from the linear constraint. To make sure I can compute the continued value $V\left(x',s'\right)$ for every result x' at each iteration, I implement a piecewise linear interpolation strategy when x' is not on the discrete grid for V.

1.5.1 Extrapolation

However, the solution for $x'\left(s_S\right)$ obtained this way may well be outside the lower and upper bounds of the grid defined for x'. To solve this issue, I use extrapolation, i.e, for a fixed $s'\in\mathcal{S}$,

$$V\left(x',s'\right)=V\left(x_N,s'\right)+\lambda^{\text{upp}}\frac{V\left(x_N,s'\right)-V\left(x_{N-1},s'\right)}{x_N-x_{N-1}}\left(x'-x_N\right)$$

and

$$V\left(x',s'\right)=V\left(x_0,s'\right)-\lambda^{\text{low}}\frac{V\left(x_1,s'\right)-V\left(x_0,s'\right)}{x_1-x_0}\left(x_0-x'\right)$$

where x_k denotes the k-th element of the grid and N is the index of the maximum element of the grid. Ideally, we set $\lambda^{\text{upp}}<1,\lambda^{\text{low}}>1$ so that we impose a penalty if x' falls outside the grid.

1.5.2 Dimension reduction

Another issue associated with solving the above problem is that now we have the control variable of length $1+S-1=S$. This may be easy if we solve the program with a few states but when S becomes large the computational resources demanded is higher for every iteration. To overcome this, note that the problem imposes some distinct structures on the solution. Each period, the continuation Ramsey problem yields the following Lagrangian

$$\begin{aligned}\mathcal{L}_1=&u\left(n-g(s),1-n\right)+\beta\sum_{s'\in\mathcal{S}}\pi\left(s'\mid s\right)V\left(x',s'\right)\\&+\Phi_1\left(x,s\right)\left(u_c\left(n-g\left(s\right),1-n\right)-u_l n+\beta\sum_{s'\in\mathcal{S}}\pi\left(s'\mid s\right)x'\left(s'\right)-x\right)\end{aligned}$$

And FOC yields

$$V_x\left(x',s'\right)=\Phi_1\left(x,s\right),\ \forall s'\in\mathcal{S}$$

Suppose V is injective. This condition implies that $x'\left(s_i\right)=x'\left(s_j\right)=x',\forall s'\in\mathcal{S}$, that is, x' is a constant vector for a given state (x,s). Under this condition, the implementability constraint yields

$$\begin{aligned}x&=u_c\left(n-g\left(s\right),1-n\right)-u_l n+\beta\sum_{s'\in\mathcal{S}}\pi\left(s'\mid s\right)x'\left(s'\right)\\&=u_c\left(n-g\left(s\right),1-n\right)-u_l n+\beta x'\end{aligned}$$

where for a given n, we can derive x'. And the choice set is now a scalar value.

Similarly, the Lagrangian for the initial recursive Ramsey problem is

$$\begin{aligned}\mathcal{L}_0=&u\left(n_0-g_0,1-n_0\right)+\beta\sum_{s_1\in\mathcal{S}}\pi\left(s_1|s_0\right)V\left(x'\left(s_1\right),s_1\right)\\&+\Phi_0\left(u_{c,0}\left(n_0-g_0,1-n_0\right)-u_{l,0}n_0+\beta\sum_{s_1\in\mathcal{S}}\pi\left(s_1|s_0\right)x'\left(s_1\right)-u_{c,0}b_0\right)\end{aligned}$$

And it follows that again

$$V_x\left(x_1', s_1'\right) = \Phi_0 \ \forall s' \in \mathcal{S}$$

And, for a given $u_{c,0} b_0$ and fix n_0, x' can be directly solved by the constraint linear in x'. In addition, an envelop condition for $t \geq 1$ is that

$$V_x\left(x, s\right) = \Phi_1$$

This and the above equations together imply

$$V_x\left(x_t, s_t\right) = \Phi_1\left(x, s\right) = \Phi_0$$

which is fixed for a given initial conditions. This phenomenon is known as the state-variable degeneracy, since this equation implies that x_t is simply a time-invariant function of s_t. This can be seen from the numerical implementation.

The dimension reduction indeed greatly improves the performance of the numerical implementation.

- When there are 4 states, the second approach takes only about 83 seconds while the first approach takes about 202 seconds.
- When there are 6 states, the second approach takes about 135 seconds while the first one takes about 352 seconds.

1.5.3 MATLAB Codes

Here, I attach my codes by section and functions.

The main program is solved in the ′time0_problem′ function. Whether to use the dimension reduction approach is taken as an input parameter REDUCE $\in \{0, 1\}$. When REDUCE $= 0$, the program proceeds with *fmincon* and otherwise it uses *fminbnd* which applies the golden search algorithm since the program becomes one-dimensional optimization over $n \in [0, 1]$.

```
function Sol = time0_problem(x0, param)
    % This function solves the time0 social planner's problem where the continuation value
    % function is given by solution in time1_problem.
    % x0 is a struct with all initial values of the model.

    % if we want to implement the dimension reduction strategy
    REDUCE = param.REDUCE;

    % unpack
    nS = param.nS;
    x_grid = param.x_grid;
    gfunc = param.gfunc;
    beta = param.beta;
    opts = param.opts;

    b0 = x0.b0;
    s0 = x0.s0;
    g0 = gfunc(s0);

    % first solve for the continuation bellman equation
    sol = time1_problem(param);

    disp('Time-0 reached.')

    % unpack solution
    Vend = sol.V;
    Vhist = sol.Vhist;
    policy_n = sol.policy_n;
    policy_x_prime = sol.policy_x_prime;

    % define objective2solve s a func of choices for a given state (b0, s0)
    objective2max = @(z) - objective0(z,b0,s0,Vend,param);

    if REDUCE==0
        % create bounds for maximization
        nbounds = [0,1];
        xbounds = [min(x_grid), max(x_grid)];
        bounds = [nbounds;repmat(xbounds,[nS-1,1])];
        UB = bounds(:,2);
        LB = bounds(:,1);

        % define initial value as solution of the time1 problem
        n0_init = policy_n(1,s0);
        x_prime0_init = policy_x_prime(:,1,s0);
        z0 = [n0_init;x_prime0_init(1:end-1)];

        % maximize
        [z_sol,fval] = fmincon(objective2max,z0,[],[],[],[],LB,UB,[],opts);
```

```matlab
    elseif REDUCE==1
        LB = 0;
        UB = 1;

        % maximize
        [z_sol,fval] = fminbnd(objective2max,LB,UB);
    end

    % get solution as policy
    if REDUCE==0
        n0 = z_sol(1);
        x_prime0_ExceptLast = z_sol(2:end);
        [MU_c0,MU_l0,~] = util(n0-g0,1-n0,param);

        x0 = MU_c0*b0;
        x_prime0_Last = find_x_prime_Last(x0,s0,z_sol,MU_c0,MU_l0,param);

        % return everything from time0 and time1 problem
        Sol.V = Vend;
        Sol.Vhist = Vhist;
        Sol.W = - fval;
        Sol.policy_n = policy_n;
        Sol.policy_x_prime = policy_x_prime;
        Sol.policy_n0 = n0;
        Sol.policy_x_prime0 = [x_prime0_ExceptLast;x_prime0_Last];

    elseif REDUCE==1
        n0 = z_sol;
        [MU_c0,MU_l0,~] = util(n0-g0,1-n0,param);
        x0 = MU_c0*b0;

        % find x_prime0
        x_prime0 = (x0 + MU_l0*n0 - MU_c0)/beta;
        Sol.V = Vend;
        Sol.Vhist = Vhist;
        Sol.W = - fval;
        Sol.policy_n = policy_n;
        Sol.policy_x_prime = policy_x_prime;
        Sol.policy_n0 = n0;
        Sol.policy_x_prime0 = repmat(x_prime0,[nS,1]);
    end

end
```

Here the time-0 objective function is defined as below. It takes state (b_0, s_0) as given instead of (x, s).

```matlab
function Val = objective0(z, b0, s0, Vend, param)
    % This function defines the objective of the initial Ramsey planner
    % state variables: b0 and s0;
    % choice variables: z including n and x_prime_ExceptLast;
    % implementability constraint yields x_prime_Last;
    % value function from the continuation ramsey problem: Vend;
    % z is the choice variable as a vector

    % check REDUCE mode
    REDUCE = param.REDUCE;

    beta = param.beta;
    Trans = param.Trans;
    gfunc = param.gfunc;
    nS = param.nS;

    if REDUCE==0
        % proceed as usual
        n0 = z(1);
        x_prime_ExceptLast = z(2:end);

        g0 = gfunc(s0);   % government spending at initial state s0
        [MU_c0, MU_l0, U] = util(n0-g0,1-n0,param);   % utility

        % get x_prime_Last from implementability constraint
        x0 = MU_c0*b0;
        x_prime_Last = find_x_prime_Last(x0,s0,z,MU_c0,MU_l0,param);
        x_prime = [x_prime_ExceptLast; x_prime_Last];

    elseif REDUCE==1
        n0 = z;
        g0 = gfunc(s0);   % government spending at initial state s0
        [MU_c0, MU_l0, U] = util(n0-g0,1-n0,param);   % utility

        x0 = MU_c0*b0;
        x_prime = (x0 + MU_l0*n0 - MU_c0)/beta;
        x_prime = repmat(x_prime,[nS,1]);
    end

    % evaluate Vend(x',s')
    Vend_Val = zeros(nS,1);

    for s_prime =1:nS
        x = x_prime(s_prime);
        Vend_Val(s_prime) = interp_extraplinear(x,s_prime,Vend,param);
    end

    % finally calculate objective
    Val = U + beta*Trans(s0,:)*Vend_Val;
end
```

The next period value function and initial values are given by solutions to the ′time1_problem′ embedded in the ′time0_problem′ as follows. The program runs the main loop with value function iteration over both state grids.

```
function sol = time1_problem(param)
    % This function solves the continuation Ramsey problem through backward induction
    % and value function iteration until period 1 given parameters set in param.

    % check REDUCE mode
    REDUCE = param.REDUCE;


    nS = param.nS;
    x_grid = param.x_grid;
    MaxIter = param.MaxIter;
    opts = param.opts;
    tol = param.tol;        % stopping threshold

    gfunc = param.gfunc;
    beta = param.beta;
    nx = length(x_grid);

    % define value functions
    Vend = zeros(nx,nS);
    Vcont = zeros(nx,nS);
    Vhist = zeros(nx,nS,MaxIter);

    % define policy functions
    policy_n = zeros(nx,nS);
    policy_x_prime = zeros(nS,nx,nS);

    % the initial value for objective2solve is n = 0 and x' = mean(x_grid) at steady state for all s';
    % then the initial value for period t iteration is the solution in period t+1; we denote
    % all choice variables z = [n;x_prime_exceptLast]

    if REDUCE==0
        z0 = [0.5; repmat(mean(x_grid),[nS-1,1])];
    elseif REDUCE==1
        z0 = 0.5;
    end
```

```

    % begin value function iteration
    for iter=1:MaxIter
        for s=1:nS
            for xk=1:length(x_grid)

                % unpack grid point
                x = x_grid(xk);

                % define objective2solve s a func of choices for a given state (x, s)
                    objective2max = @(z) - objective(z, x, s, Vend, param);

                if REDUCE==0
                    % create bounds for maximization
                    nbounds = [0,1];
                    xbounds = [min(x_grid), max(x_grid)];
                    bounds = [nbounds;repmat(xbounds,[nS-1,1])];
                    UB = bounds(:,2);
                    LB = bounds(:,1);

                    % update initial policy as optimum from last iteration
                    if iter>1
                        n0 = policy_n(xk,s);
                        x_prime0 = policy_x_prime(:,xk,s);
                        z0 = [n0;x_prime0(1:end-1)];
                    end

                    % maximize
                    [z_sol,fval] = fmincon(objective2max,z0,[],[],[],[],LB,UB,[],opts);

                elseif REDUCE==1
                    UB = 1;
                    LB = 0;
                    [z_sol,fval] = fminbnd(objective2max,LB,UB);
                end

                Vcont(xk,s) = -fval;

                if REDUCE==0
                    n = z_sol(1);
                    g = gfunc(s);
                    [MU_c, MU_l, ~] = util(n-g,1-n,param);    % utility

                    x_prime_Last = find_x_prime_Last(x,s,z_sol,MU_c,MU_l,param);

                    % pack policy function
                    policy_n(xk,s) = n;
                    policy_x_prime(:,xk,s) = [z_sol(2:end);x_prime_Last];

                elseif  REDUCE==1
                    n = z_sol;
                    g = gfunc(s);
                    [MU_c, MU_l, ~] = util(n-g,1-n,param);    % utility
                    x_prime = (x + MU_l*n - MU_c)/beta;
```

```
                    % pack policy function
                    policy_n(xk,s) = n;
                    policy_x_prime(:,xk,s) = repmat(x_prime,[nS,1]);
                end
            end
        end
        % report progress
        dist = max(max(abs(Vcont-Vend)));
        fprintf('Time-1 progress: iter=%d, dist=%f\n', iter, dist)

        % stopping criterion
        if dist < tol
            break;
        end

        Vend = Vcont;
        Vhist(:,:,iter) = Vcont;
    end

    sol.V = Vcont;
    sol.policy_n = policy_n;
    sol.policy_x_prime = policy_x_prime;
    sol.Vhist = Vhist;
end

```

Here, the objective function is similar to $'$objective0$'$ but with states (x, s) as given. And it again accommodates both solution methods by taking z as either a one-dimensional or S dimensional object.

```
function Val = objective(z, x, s, Vend, param)
    % This function defines the objective of the continuation Ramsey planner
    % state variables: x and s;
    % choice variables: z including n and x_prime_ExceptLast;
    % implementability constraint yields x_prime_Last;
    % value function from the last iteration: Vend;

    % z is the choice variable as a vector

    % check REDUCE mode
    REDUCE = param.REDUCE;

    beta = param.beta;
    Trans = param.Trans;
    gfunc = param.gfunc;
    x_grid = param.x_grid;
    nS = param.nS;


    if REDUCE==0
        n = z(1);
        x_prime_ExceptLast = z(2:end);

        g = gfunc(s);   % government spending at state s
        [MU_c, MU_l, U] = util(n-g, 1-n,param);   % utility

        % get x_prime_Last from implementability constraint
        x_prime_Last = find_x_prime_Last(x,s,z,MU_c,MU_l,param);
        x_prime = [x_prime_ExceptLast; x_prime_Last];

    elseif REDUCE==1
        n = z;
        g = gfunc(s);
        [MU_c, MU_l, U] = util(n-g, 1-n,param);   % utility

        x_prime = (x + MU_l*n - MU_c)/beta;
        x_prime = repmat(x_prime,[nS,1]);
    end


    % evaluate Vend(x',s')
    Vend_Val = zeros(nS,1);

    for s_prime =1:nS
        x = x_prime(s_prime);
        Vend_Val(s_prime) = interp_extraplinear(x,s_prime,Vend,param);
    end

    % finally calculate objective
    Val = U + beta*Trans(s,:)*Vend_Val;
end
```

A few additional functions make the program more convenient. $'$find_x_prime_last$'$ finds $x'(s_{end})$ using the implementability constraint linear in $x'(s_1 : s_{end-1})$, which reduces one degree of freedom of the choice set. The utility function is specified to be log form separate-additive. And $'$interp_extraplinear$'$ implements the interpolation and extrapolation strategy to obtain value function outcomes both on and off the x_grid. The interpolation is piecewise linear and extrapolation follows the above penalty routine.

```
function x_prime_Last = find_x_prime_Last(x,s,z,MU_c,MU_l,param)
    % calculates the last element of x' from the implementability constraint
    Trans = param.Trans;
    beta = param.beta;

    % unpack z
    n = z(1);
    x_prime_ExceptLast = z(2:end);

    % subtract RHS from LHS
    LminusR = x - MU_c + MU_l*n - beta*Trans(s,1:end-1)*x_prime_ExceptLast;
    x_prime_Last = LminusR/(beta*Trans(s,end));
end


function [MU_c, MU_l, U] = util(c, l, param)
    % compute utility, marginal utility of consumption and marginal utility of labor
    gamma = param.gamma;
    if c<0
        U = -1e9;
    else
        U = log(c) + gamma*log(l);
    end
    MU_c = 1/c;
    MU_l = gamma/l;
end


function Val = interp_extraplinear(x,s_prime,Vend,param)
    % perform a linear interpolation
    Vend_s = Vend(:,s_prime);

    x_grid = param.x_grid;
    lambda_upp = param.lambda_upp;  % extrapolation penalty
    lambda_low = param.lambda_low;

    slope_upp = (Vend_s(end) - Vend_s(end-1)) / (x_grid(end) - x_grid(end-1));
    slope_low = (Vend_s(2) - Vend_s(1)) / (x_grid(2) - x_grid(1));

    % interpolation
    if x>=min(x_grid) & x<=max(x_grid)
        i = lookup(x_grid,x,3);
        weightL = (x-x_grid(i)) / (x_grid(i+1)-x_grid(i));
        % evaluate Val as weighted sum of two endpoints
        Val = (1-weightL)*Vend_s(i) + weightL*Vend_s(i+1);

    elseif x>max(x_grid)            % extrapolation
        Val = Vend_s(end) + lambda_upp*slope_upp*(x-max(x_grid));

    elseif x<min(x_grid)
        Val = Vend_s(1) - lambda_low*slope_low*(min(x_grid)-x);
    end
end
```

1.5.4 Example outcome

The outcome is stored in a struct as follows:

```
outcome =

  struct with fields:

                V: [200×4 double]
            Vhist: [200×4×500 double]
                W: -76.4606
         policy_n: [200×4 double]
   policy_x_prime: [4×200×4 double]
        policy_n0: 0.6294
  policy_x_prime0: [4×1 double]
```

The optimal value function for the continuation problem is indeed an invariant function of the exogenous state s, a manifestation of the state variable degeneracy.

```
>> outcome.V

ans =

  -76.4606  -76.5535  -77.1316  -76.5332
  -76.4606  -76.5535  -77.1316  -76.5332
  -76.4606  -76.5535  -77.1316  -76.5332
  -76.4606  -76.5535  -77.1316  -76.5332
  -76.4606  -76.5535  -77.1316  -76.5332
  -76.4606  -76.5535  -77.1316  -76.5332
  -76.4606  -76.5535  -77.1316  -76.5332
```

And for a given state (x, s), the continuation policy function $x'(s')$ is the same for all s', as shown in the FOC above. This also justifies our use of the dimension reduction approach.

```
>> outcome.policy_x_prime(2,2,:)

ans(:,:,1) =

   -5.7649


ans(:,:,2) =

   -5.7651


ans(:,:,3) =

   -5.7652


ans(:,:,4) =

   -5.7651
```

2 References

Ljungqvist, Lars & Sargent, Thomas J., 2018. "Recursive Macroeconomic Theory, Third Edition," MIT Press Books, The MIT Press.