A Numerical Solution to "Stairway to heaven or highway to hell: Liquidity, sweat equity, and the uncertain path to ownership"*

23/04/2023

1 Overview

In this project, we consider a numerical example of the problem discussed in the paper "Stairway to heaven or highway to hell: Liquidity, sweat equity, and the uncertain path to Ownership" written by Krishna, Lopomo, and Taylor. As the main problem is formulated as Dynamic programming, in this report, we would like to use value function iteration to give a numerical solution to this problem.

2 A Review of the Theory

2.1 Key elements of the Model

There is a principal and an agent; both have risk-natural, time-separable utility. At the infinite, discrete-time horizon $t = 1, 2, 3, ..., N$,

- the principal will buy (contractable) q_t unit of good from the agent with payment p_t, with utility

$$U_t^P = R(q_t) - p_t$$

- the agent will receive payment p_t and produce q_t, with private marginal cost $\theta_t \in \{\theta_1, \theta_2 | \theta_1 < \theta_2\}$ and drawn independently with probability $f_i \equiv \Pr[\theta_t = \theta_i] > 0$.

$$U_t^A = p_t - \theta_t q_t$$

Two frictions are considered in the Model:

1. the realization of θ_t is privately shown to the agent but not the principal.
2. Liquidity constrain: $U_t^A \geqslant 0$

*This is a report as the final essay of BA912-Dynamic Programming and Optimal Control, written by Zhiyuan Jia. I thank Prof. Pino Lopomo for the proposal of this project. He also helps a lot to complete it.

Notice that if either of the frictions/constraints is not included, then **First Best** will be achieved:

$$v^{\mathrm{FB}} := \frac{1}{1-\delta}\left[f_1\left[R\left(q_1^*\right)-\theta_1 q_1^*\right]+f_2\left[R\left(q_2^*\right)-\theta_2 q_2^*\right]\right], \quad \text{where} \quad R'\left(q_i^{\mathrm{FB}}\right)=\theta_i \tag{FB}$$

2.2 Contract

Now using the mechanism design approaches to deal with the Model with frictions. First, we restricted to consider the case in which the agent honestly reported the information (θ_i) with incentive-compatible constrain (Revelation Principle). Secondly, we can further restrict to a recursive mechanism; information per stage can be summarized by a sufficient statistic: v, **the agent's lifetime promised expected utility** because there is a recursive structure on the mechanism. Therefore, drop the time subscript, and v will be used as the state variable for the current period and denote w_i as the state starting the next period. If define the agent's instantaneous rent as $u_i = p_i - \theta_i q_i$, then **the principal's discounted expected utility** under an optimal contract (u, q, w), is represented by a concave and continuously differentiable function $P(v)$ generated from the following dynamic programming [1]

Problem $\mathcal{P}$

$$P(v) = \max_{u_i,q_i,w_i} \sum_{i=1,2} f_i\left[R\left(q_i\right)-\theta_i q_i - u_i + \delta P\left(w_i\right)\right] \tag{OBJFN}$$

s. to:

$$f_1\left(u_1+\delta w_1\right)+f_2\left(u_2+\delta w_2\right)=v \tag{PK}$$

$$\forall i,j=1,2: \quad u_i+\delta w_i \geqslant u_j+\delta w_j+\left(\theta_j-\theta_i\right)q_j \tag{IC}$$

$$\forall i=1,2: \quad u_i \geqslant 0 \quad w_i \geqslant 0 \quad q_i \geqslant 0 \tag{LF}$$

To interpret the problem:

1. (OBJFN): the maximization of current (principal's) utility and discounted future expectation
2. (PK): promise-keeping, the agent's lifetime expected payoff is composed of his expected payoff in the current period and discounted future
3. (IC): incentive compatibility, in which the agent will not deviate from reporting the truth
4. (LF): liquidity constrain and feasibility

3 Numerical Example

First setting the parameters in the model:

$$R(), f_1, f_2, \theta_1, \theta_2, \delta$$

[1] See the original paper, Theorem 1, for the detailed proof

```
%% set parameters

f_1=1/2; % distribution of cost
f_2=1/2;
theta_1=1; % marginal cost
theta_2=2;
d=0.9; % discount rate per-stage
```

3.1 First best solution

Following equation (FB), we will get $q_1^{\text{FB}} = 0.2500, q_2^{\text{FB}} = 0.0625, v^{\text{FB}} = 1.8750$

```
%% First Best Solution

qFB_1=1/(4*(theta_1)^2)
qFB_2=1/(4*(theta_2)^2)
vFB=1/(1-d)*(f_1*(sqrt(qFB_1)-theta_1*qFB_1)+f_2*(sqrt(qFB_2)-theta_2*qFB_2))
```

3.2 Value iteration for second best

Recall that the dynamic programming problem $\mathcal{P}$ describes the optimal contract. Now we consider value function iteration to solve the problem computationally. The main difficulty is that this is a functional expression with continuous state space. Fortunately, the properties of the programming can be preserved when a finite subset of state space is considered.

However, with this simplification, we need to solve a mixed optimization problem with w_i taking discrete values in each update, but c_i and q_i will be taken from continuous space. A natural way to simplify is that we also consider the discrete subspace of those parameters. Another advantage of this brute-force method is that constraints can be handled easily.

Step 1: Create gridV, gridQ, and gridU.

Notice that every grid needs to be created on a bounded interval. According to the theory of first best result (which will create the largest production), we do gridV on

$$V = [0, v^{FB}]$$

and do gridQ on

$$Q = [0, q_1^{FB}]$$

For gridU, we take it on

$$U = [0, u_{max}]^{2}$$

[2] We take $u_{max} = 3$; a large enough choice

```
%% Create a grid for v

vmin = 0; % minimum, maximum vFB
numv = 19; % grid points + 1
% grid = (vFB-vmin)/vgrid; % grid
vgrid = vmin:(vFB-vmin)/numv:vFB;
vgrid = vgrid';
[Nv,n] = size(vgrid);


%% Also Create the Grid for q, u
qmin=0;
numq=19;
qgrid= qmin:(qFB_1-qmin)/numq:qFB_1;
qgrid=qgrid';
[Nq,n]=size(qgrid);


umin=0;
umax=3;
numu=19;
ugrid= umin:(umax-umin)/numu:umax;
ugrid=ugrid';
[Nu,n]=size(ugrid);
```

Step 2: Define a function P_0 that maps from the gridV to the interval

```
%% Define a Function P(0), intialized as all zero function

p0 = zeros(Nv,1);
```

Step 3: Update the function recursively

Specifically, For $v \in$ gridV, solve the problem

$$P_1(v) = \max_{u_i,q_i,w_i} \sum_{i=1,2} f_i \left[R\left(q_i\right) - \theta_i q_i - u_i + \delta P_0\left(w_i\right)\right]$$

s.t. (PK), (IC), (LF) and $w_1, w_2 \in$ gridV, $q_1, q_2 \in$ gridQ, $u_1 \in$[3] gridU.

Then solve the problem

$$P_2(v) = \max_{u_i,q_i,w_i} \sum_{i=1,2} f_i \left[R\left(q_i\right) - \theta_i q_i - u_i + \delta P_1\left(w_i\right)\right]$$

s.t. (PK), (IC), (LF) and $w_1, w_2 \in$ gridV, $q_1, q_2 \in$ gridQ. $u_1 \in$ gridU.

Recursively solve for P_3, P_4....

[3] Notice u_2 is determined by (PK) in this procedure

```
%% Update the P function throughout min
tol = 0.001;
maxits = 300;
dif = tol + 1000; % P_1 - P_0
numOfIterations = 0;

while dif > tol && numOfIterations < maxits
    for i = 1:Nv
        v0 = vgrid(i,1);
        p1cand = -999; % intialized a number for each data point;
        % Do bruteforce checking; go through u_1,w_1,w_2,q_1,q_2
        for i_1 = 1:Nv
            w_1 = vgrid(i_1,1);
            pw_1=p0(i_1,1);
            for i_2 = 1:Nv
                w_2 = vgrid(i_2,1);
                pw_2 = p0(i_2,1);
                for j_1=1:Nq
                    q_1=qgrid(j_1,1);
                    for j_2=1:Nq
                        q_2=qgrid(j_2,1);
                        for l_1=1:Nu
                            u_1=ugrid(l_1,1);
                            u_2=2*v0-u_1-d*(w_1+w_2); %u_2 defined by (PK)
                            if u_1<0 ...
                                    || u_2<0 ...
                                    || u_1+d*w_1 < u_2+d*w_2+(theta_2-theta_1)*q_2 ...
                                    || u_2+d*w_2 < u_1+d*w_1+(theta_1-theta_2)*q_1
                                va = -99999999999;
                            else
                                va = f_1*(sqrt(q_1)-theta_1*q_1-u_1 + d*pw_1)...
                                    +f_2*(sqrt(q_2)-theta_2*q_2-u_2 + d*pw_2);
                            end
                            if va>p1cand
                                p1cand=va;
                                u1cand=u_1;
                                w1cand=w_1;
                                q1cand=q_1;
                            end
                        end
                    end
                end
            end
        end
        p1(i,1) = p1cand; %update the value function for each grid point i.e. optimal ...
            value for the programming per period)
        u1(i,1) = u1cand; %update the policy
        w1(i,1) = w1cand;
        q1(i,1) = q1cand;
    end
    dif = norm(p1-p0)
    p0 = p1;
    numOfIterations = numOfIterations+1
end
```

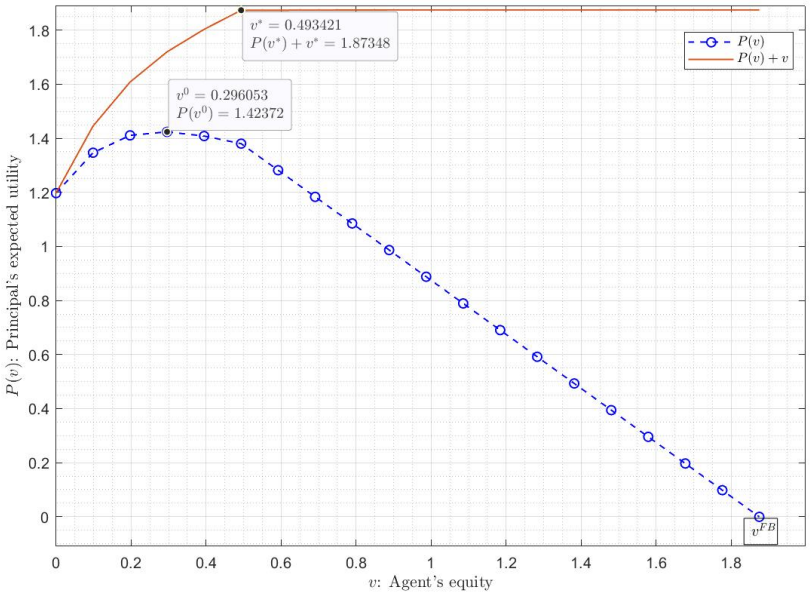

Figure 1: Value function (expected utility of the principal)

3.3 Analysis on the result

Now we use the example to check our main result in the paper.

1. In figure 1, we plot $(v, P(v))$ and $(v, P(v) + v)$. $P(v)$ is maximized at v_0, a level lower than the social optimal point v^*. After v^*, efficiency is always achieved.

2. In figure 2, we plot (v, u_i) and $(v, P(v) + v)$. Before v^* is achieved, $u_1 = 0$, no current utility is given: The optimal contract incentivizes the agent exclusively via promised future payments before he becomes a fully vested partner. Also, liquidity constraints are ameliorated as the firm grows.

3. In figure 3, we plot (v, w_i). The optimal policy is stable either at 0 (hell) or grows to the point where the agent becomes a fully vested partner (heaven).

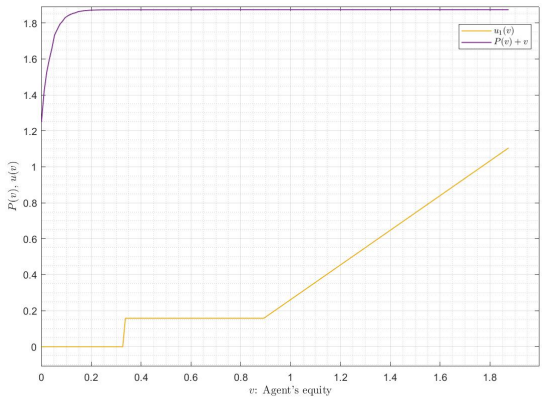

Figure 2: Instantaneous utility

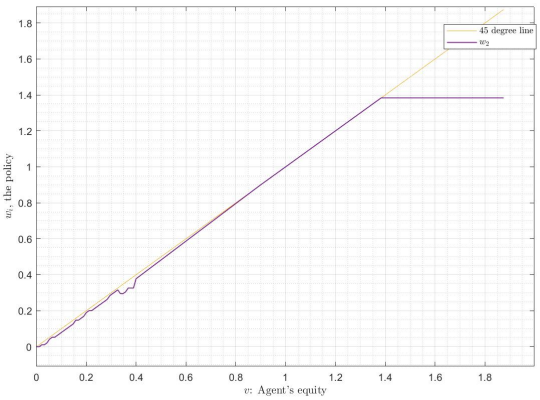

Figure 3: Policy

3.4 Modification

From both the theory and the numerical result, we know that $P(v)$ is linear when $v > v^*$, the social-optimal level. Therefore, we can reformulate the grid of the state with a nonuniform setup:

```
%% vgrid
    if 0 % uniform grid for v
        vmin = 0; % minimum, maximum vFB
        numv = 30; % grid points + 1
        % grid = (vFB-vmin)/vgrid; % grid
        vgrid = vmin:(vFB-vmin)/numv:vFB;
    end


    if 1   % nonuniform grid for v
        lower_vGrid = linspace(0, .4, 39);
        upper_vGrid = linspace(.41, vFB, 4);
        vgrid = [lower_vGrid upper_vGrid ];
    end

    vgrid = vgrid';
    Nv = size(vgrid,1);
end
```

4 Conclusion and Discussion

Although restricted to the finite state space, the numerical simulation successfully recovers the value function curve and verifies the paper's main results.

Other modifications can be added to the simulation algorithm. For example, the instantaneous variables (u_i, q_i) can be done in a continuous space with an optimization toolbox; On the other hand, linear interpolation could be done on v. Those techniques will reduce the error, but the result in this project has well characterized the solution.