VFI and three algorithms of optimization

Guanyi Yang

Colorado College

November 1, 2024

Plan

Start with

We will start with the infinite horizon HH consumption-savings model

End with

We will end this session with a comparison of three computation algorithms: grid search, GSS, EGM

Sequential format of HH c-s model

$$\begin{aligned}
&\max_{\{c_t, s_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u(c_t) \\
&\text{s.t.} \\
c_t + s_{t+1} &= w + (1+r)s_t, \ \text{t} = 0, 1, \ldots \\
s_{t+1} &\geq 0, \ \text{t} = 0, 1, \ldots \\
c_t &\geq 0, \ \text{t} = 0, 1, \ldots \\
&s_0 \text{ given}
\end{aligned} \tag{1}$$

At any given time, we are maximizing the (infinite) lifetime utility. Every period, we need to make the same decision again and again, as shown by the budget constraint.
This format of describing the infinite horizon problem is called - sequential format.

Recursive format of Optimal growth model

Sequential format helps us intuitively see what is happening in an infinite horizon. However, it is impossible to directly solve a model in sequential format with infinite horizon.

We represent it in a *recursive* format:

$$
\begin{aligned}
V(s) &= \max_{c,s'}\{u(c) + \beta V(s')\} \\
&\text{s.t.} \\
c + s' &= w + (1+r)s \\
s' &\geq 0 \\
c &\geq 0 \\
s &\text{ given}
\end{aligned}
\tag{2}
$$

Value function at any given point in time is **exactly the same** V, just with different asset s and s'. Only today and tomorrow matter.

Method of successive approximations

Algorithm for solving the model in stationary equilibrium

1. Our purpose is to find the choice of s' that makes the lifetime value the highest, and we store the value of s' in decision rule G.
2. However, we don't know what future $V(s')$ is. So we assume it a starting value of 0, and take it as given to find s' to maximize $Tv = u(c) + beta*v$ with the assumed v.
3. At the end of each iteration, we update our assumption of V (future value), since we know in steady state (stationary equilibrium), $Tv = V$.
4. We successfully found our solution when $Tv = V$. This is done by updating the "distance" of our stopping criteria.

setup

Setup before proceeding:

- utility function: $u(c) = log(c)$
- parameters: $\beta = 0.96$, $r = 0.01$ and $w = 1$.
- Grid space for state variable s: $sgrid = linspace(5, 10, 500)$;

Method 1: Grid search

Pseudo-code

1. Initiate all necessary values for solving the model: current period value: Tv, future period value v, decision rule G.
2. Define stopping rule and iteration counter for while-loop:

$$precision = 1e - 5;$$

$$distance = 2 * precision;$$

$$iteration = 0;$$

3. Initialize all possible candidates of lifetime value, name it $Tv0 = zeros(snum, snum)$; and the associated possible current consumption, $c0 = Tv0$;.

Discrete state-space dynamic programming
Pseudo-code

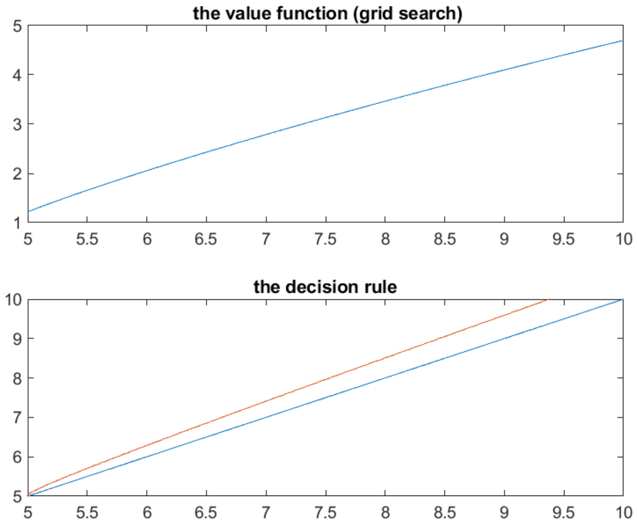
4. Use the initialized future period value v as our starting guess of future value
5. for each current s_i and each possible future s_j, fill-in the corresponding possible consumption, and store it at $c0(i, j)$.
6. for each possible consumption, find possible lifetime value according to value function $U(c) + \beta v$, and store it at the corresponding location $Tv0(i, j)$.

Discrete state-space dynamic programming

Pseudo-code

7. for each current s_i, find the value of the $Tv0(i,:)$ that returns the max value, store the value at the corresponding location $Tv(i)$, and save the location *loc*.
8. use *loc* to find the corresponding s' from *sgrid* that returns the above optimized Tv, and save it at $G(i)$.
9. Now we have just completed one iteration of the "approximation". Let's update the stopping rule $distance = max(max((abs(Tv - v))))$;.
10. We then update our initial guess of future v by our current calculation $v = Tv$;.
11. print the process, so we know how it is doing:
    ```
    sss = sprintf ( ' iteration %4d ||Tv-v|| = %8.6f
    ', iteration, distance);
    disp(sss)
    ```
12. update the iteration counter $iteration = iteration + 1$;
13. repeat from Step 4 until converges ($Tv = v$)

Result



the value function (grid search)
5
4
3
2
1
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10
the decision rule
10
9
8
7
6
5
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10

Note

- Benefit: simple and reliable
- Problem: precision, issue of non-linearity, off-grid values, curse of dimensionality

Method 2: Golden Section Search (GSS)

Pseudo-code

3. for each value of current s, we use GSS to directly find the optimal future s' and the associated value Tv.
4. Now we have just completed one iteration of the "approximation". Let's update the stopping rule $distance = max(max((abs(Tv - v))));$.
5. We then update our initial guess of future v by our current calculation $v = Tv;$.
6. print the process, so we know how it is doing:
   ```
   sss = sprintf ( ' iteration %4d ||Tv-v|| = %8.6f
   ', iteration, distance);
   disp(sss)
   ```
7. update the iteration counter $iteration = iteration + 1;$
8. repeat from Step 3 until converges ($Tv = v$)

Golden Section Search

Pseudo-code

1. given each current s, start with guessing the boundary of future s': lowest value a and highest value b. This is also where you specify the borrowing limit.
2. Golden section search uses the golden ratio ($rr = (3 - sqrt(5))/2;$) to update the trial of s'.
3. we start by using the golden ratio to calculate c ($cc = (1 - rr) * aa + rr * bb;$) and d ($dd = rr * aa + (1 - rr) * bb;$) based on the values of a and b. These are potential values of s' to try.
4. after we specify c and d, we update the value function accordingly, using c and d as next period s', and store the *negative* of the values as f_c and f_d. Make sure to interpolate future value $V(s')$ based on c and d.

Algorithm to update the search for optimal s':

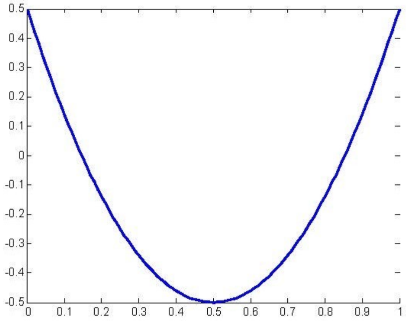
Pseudo-code

```
1. if $f_c \geq f_d$, we update the sampling $a, c, d$ and the values
   according to:
   $aa = cc;$
   $cc = dd;$
   $fc = fd;$
   $dd = rr * aa + (1 - rr) * bb;$
   and recalculate $f_d$ with the new $d$.
2. if $f_c < f_d$, we update the sampling $b, c, d$ and the values
   according to:
   $bb = dd;$
   $dd = cc;$
   $fd = fc;$
   $cc = (1 - rr) * aa + rr * bb;$
   and recalculate $f_c$ with the new $c$.
3. iterate until $a, b, c, d$ are basically the same
   $(abs(dd - cc) < 1e - 8)$
```

GSS search example

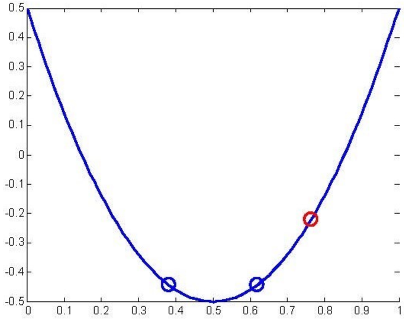
$$f(x) = -4x(1-x) + \frac{1}{2}$$

(a, b) = (0.0000, 1.0000) (c, d) = (0.3820, 0.6180) (fc,fd) = (-0.4443, -0.4443)



GSS search example

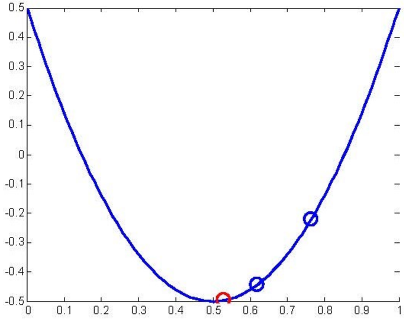
$f_c \geq f_d$ (blue circles) and c becomes a, sample d'



(a, b) = (0.3820, 1.0000) (c, d) = (0.6180, 0.7639) (z, fval) = (0.7639, -0.2214)

GSS search example

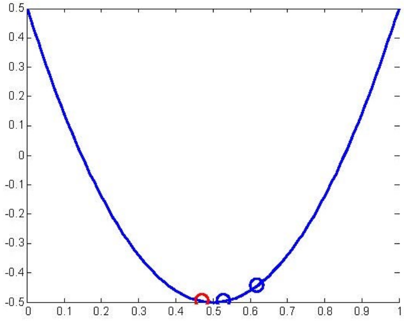
$f_c < f_d$ (blue circles) and d becomes b, sample c'



(a, b) = (0.3820, 0.7639) (c, d) = (0.5279, 0.6180) (z, fval) = (0.5279, -0.4969)

GSS search example

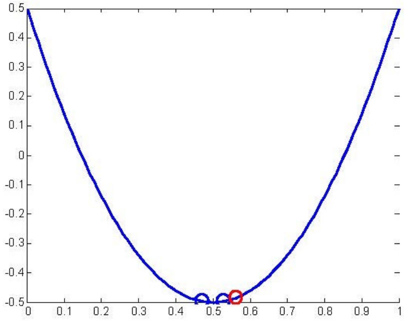
$f_c < f_d$ (blue circles) and d becomes b, sample c'



(a, b) = (0.3820, 0.6180) (c, d) = (0.4721, 0.5279) (z, fval) = (0.4721, -0.4969)

GSS search example

$f_c \geq f_d$ (blue circles) and c becomes a, sample d'



(a, b) = (0.3820, 0.6180) (c, d) = (0.4721, 0.5279) (z, fval) = (0.4721, -0.4969)

Interpolation (of future value based on each guessed s' on $sgrid$

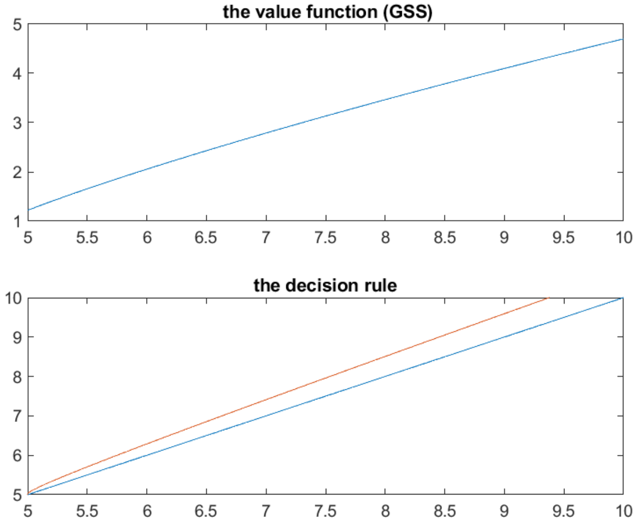
Pseudo-code

1. find where s' is on $sgrid$, label it $sindex$ as the grid point that is immediately lower than s' and $sindex + 1$ as the grid point that is immediately higher than s'.
2. assign weight $\frac{sgrid(sindex+1)-s'}{(sgrid(sindex+1)-sgrid(sindex))}$; to be the downward weight to $sindex$ and $1 - weight$ to be the upward weight to $sindex + 1$.
3. the interpolated future value based on s' on $sgrid$ is: $weight * V(sindex) + (1 - weight) * V(sindex + 1)$

Important note:

- GSS searches for the global minimum of a quasiconvex function. Value function optimization is a global maximum for a concave function. So, we add a $-$ sign for the value function during GSS. Remember to switch it back after the end.
- Make sure to set search boundaries at the initial setup of a and b, incorporating borrowing limit and cash-on-hand (or conditions alike).
- benefit: preserves non-linearity (with spline interpolation), accuracy with sparse grid, reliable
- issues: curse of dimensionality

Result



the value function (GSS)
5
4
3
2
1
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10
the decision rule
10
9
8
7
6
5
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10

Method 3: Endogenous Grid Method (EGM)

- The reason we use EGM: for speed gain
- The major issue: non-monotonicity/non-convexity (generalized GEM) and multiple endogenous dimensions (G^2EGM or EGM for multiple dimensions)
- The core idea: assume grid for future s', and endogenously back-out current s to avoid maximization routine.

Method 3: Endogenous Grid Method (EGM)

Pseudo-code

1. Initiate all necessary values for solving the model: current period value: Tv, future period value v, exogenous grid for decision rule *sgrid*. Know that this is for s' rather than s.
2. Define stopping rule and iteration counter for while-loop:

$$precision = 1e-5;$$
$$distance = 2 * precision;$$
$$iteration = 0;$$

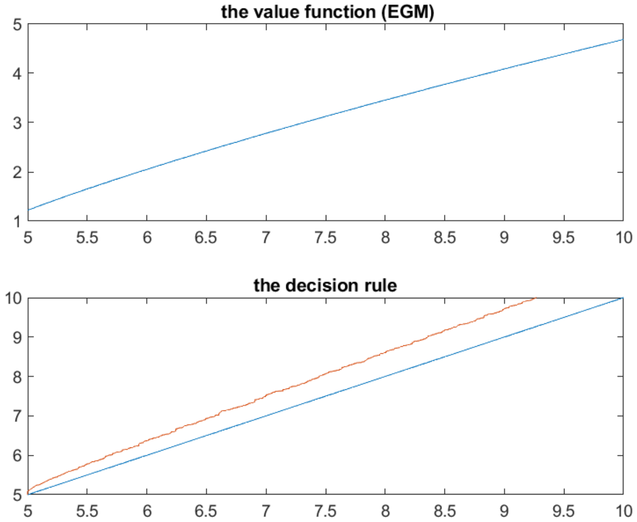
3. for each value of future s' (value on the *sgrid*), we use the Euler equation and budget constraint to back out current s, which creates an *endogenous grid*, name it *segrid*.
4. in relation to each value on the endogenous grid, each value on the exogenous grid *sgrid* is a savings rule. So, we interpolate the exogenous grid *sgrid* on the endogenous grid *segrid* to identify the "off"-grid savings rule.

Method 3: Endogenous Grid Method (EGM) - continue

Pseudo-code

5. we now interpolate each future value $V(s')$ based on the exogenous grid *sgrid* and savings rule s'.
6. Now we have just completed one iteration of the "approximation". Let's update the stopping rule $distance = max(max((abs(Tv - v))));$.
7. We then update our initial guess of future v by our current calculation $v = Tv;$.
8. print the process, so we know how it is doing:
   ```
   sss = sprintf ( ' iteration %4d ||Tv-v|| = %8.6f
   ', iteration, distance);
   disp(sss)
   ```
9. update the iteration counter $iteration = iteration + 1;$
10. repeat from Step 3 until converges ($Tv = v$)

Result



the value function (EGM)
5
4
3
2
1
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10
the decision rule
10
9
8
7
6
5
5
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10