Advanced Macroeconomics II

Handout 2 - Dynamic Programming, VFI+

Sergio Ocampo

Western University

September 23, 2020

What does a typical problem look like?

1. A dynamic programming problem with:
   - At least two choice variables (c, ℓ)
   - Two to four continuous state variables $(a/k, h, \epsilon, z)$
   - At least two discrete state variables (age, occupation)
   - Non-concavities (fixed costs, adjustment costs, asymmetries)
2. Part of a general equilibrium environment
   - At least two prices (r, w) solved as function of aggregate state
   - Keep track of distribution of agents
   - Potentially aggregate shocks (considerably harder)
3. Estimate/Calibrate 5-15 parameters
   - No analytical solution for moments
   - Non-smooth objective function

What does a typical problem look like?

1. A dynamic programming problem with:
   - At least two choice variables (c, ℓ)
   - Two to four continuous state variables $(a/k, h, \epsilon, z)$
   - At least two discrete state variables (age, occupation)
   - Non-concavities (fixed costs, adjustment costs, asymmetries)
2. Part of a general equilibrium environment
   - At least two prices (r, w) solved as function of aggregate state
   - Keep track of distribution of agents
   - Potentially aggregate shocks (considerably harder)
3. Estimate/Calibrate 5-15 parameters
   - No analytical solution for moments
   - Non-smooth objective function

Dynamic programming

Prototypical DP problem:

$$V(k,z) = \max_{\{c,k'\}} u(c) + \beta E\left[V\left(k',z'\right)|z\right]$$
$$\text{s.t.} c + k' = f(k,z)$$
$$z' = h(z,\eta);\eta \text{ stochastic}$$

- Useful in representative and heterogeneous agent problems
- What constitutes a solution?
  - Value function ($\mathbf{V}$) and policy functions ($\mathbf{g^c}, \mathbf{g^k}$)

Dynamic programming PROBLEMS

1. We are looking for functions $\mathbf{V}$ and $\mathbf{g^c}, \mathbf{g^k}$

$$\mathbf{V}(k, z) = \max_{\{\mathbf{c}, \mathbf{k}'\}} u(c) + \beta E\left[V\left(k', z'\right) |z\right]$$
$$\text{s.t.} c + k' = f(k, z)$$
$$z' = h(z, \eta); \eta \text{ stochastic}$$

- Functions are infinite-dimensional objects... unclear how to find them

Dynamic programming PROBLEMS

2 The problem involves solving a maximization

$$V(k,z) = \max_{\{c,k'\}} u(c) + \beta E\left[\mathbf{V}\left(\mathbf{k}',\mathbf{z}'\right)|z\right]$$

$$\text{s.t.} c + k' = \mathbf{f}(\mathbf{k},\mathbf{z})$$

$$z' = h(z,\eta); \eta \text{ stochastic}$$

- Maximization depends on the solution to the problem!
- Control variables can be continuous (hard... we need derivatives)
- Control variables can be discrete (also hard... no derivatives)
- Choice set can be non-convex

Dynamic programming PROBLEMS

3 The problem involves taking expectations

$$V(k,z) = \max_{\{c,k'\}} u(c) + \beta \mathbf{E}\left[\mathbf{V}\left(\mathbf{k}', \mathbf{z}'\right) | \mathbf{z}\right]$$
$$\text{s.t.} c + k' = f(k,z)$$
$$z' = h(z,\eta); \eta \text{ stochastic}$$

- Expectation is over the solution of the problem!
- Expectations are hard... they involve integrals... integrals are the worst

Importance of analytical results

- How do you know if there is a (unique) solution to your problem?
- What do you know about how your solution looks like?
  - Monotone? Increasing? Concave? Linear?
- Answers help you find good initial conditions
  - Key for stability and speed of numerical methods
- Answers let you contrast numerical solution to predictions
  - How do you know if you found the right answer?

Contraction mappings - Quick review

Contraction Mapping: Let (S, d) be a metric space and $T : S \to S$ be a mapping of S into itself. T is a contraction with modulus β, if for some $\beta \in (0, 1)$ we have:

$$\forall_{v_1, v_2 \in S} \quad d\left(Tv_1, Tv_2\right) \leq \beta d\left(v_1, v_2\right)$$

- Turns out the DP problem above defines a contraction on the space of functions (verify with Blackwell's sufficient conditions)

$$Tv\left(k, z\right) = \max_{\left\{c, k'\right\}} u\left(c\right) + \beta \mathbf{E}\left[\mathbf{v}\left(\mathbf{k}', \mathbf{z}'\right)|\mathbf{z}\right]$$
$$\text{s.t.}\, c + k' = f\left(k, z\right)$$
$$z' = h\left(z, \eta\right); \eta \text{ stochastic}$$

- Solution to DP problem is a fixed point of the contraction: $V = \mathbf{T}V$

Contraction mapping theorem

Turns out all contractions have a unique fixed point!

Contraction Mapping Theorem: Let (S, d) be a **complete** metric space and $T : S \to S$ a contraction mapping on S. Then, T has a unique fixed point $v^\star \in S$ such that:

$$\forall_{v_0 \in S} \quad v^\star = Tv^\star = \lim_{n \to \infty} T^n v_0$$

The CMT is the best result you can ever hope for

1. Gives you a solution
2. Gives you a unique solution
3. Gives you an algorithm that converges globally

But it gets better!

Contraction mapping corollary

Corollary - Contraction Mapping Theorem: Let (S, d) be a complete metric space, $T : S \to S$ a contraction mapping on S and $v^\star$ the fixed point of T on S.

- If $\overline{S}$ is a closed subset of S, and $T(\overline{S}) \subset \overline{S}$, then $v^\star \in \overline{S}$.
- If in addition there is a set $\tilde{S}$ such that $T(\overline{S}) \subset \tilde{S} \subset \overline{S}$, then $v^\star \in \tilde{S}$.

The corollary lets us apply the CMT to non-complete spaces

- S can be the space of continuous, bounded functions
- $\overline{S}$ can add weak concavity
- $\tilde{S}$ can add strict concavity

Analytical solution

Some problems can be solved analytically

1. Guess and verify
2. Manual VFI or backwards induction (finite horizon)
3. Euler equations

Very limited in practice

- Very few problems can be solved this way
  - Exceptions: Angeletos (2007), Moll (2014), Itskhoki & Moll (2019), Achoud, et al (2020), Benhabib, Bisin (2018), Akira Toda, et al (2019)
- Euler equations still useful - Reduce problem
- Problems provide good initial conditions

Analytical solution: Guess and verify

$$V(k) = \max_{\{c,k'\}} \log(c) + \beta V\left(k^{'}\right) \qquad \text{s.t. } c + k^{'} = zk^{\alpha}$$

Guess and verify (problem set): $V(k) = a_0 + a_1 \log k$

1. Get Euler equation given guess.
2. Solve for policy function given guess.
3. Replace back and solve for coefficients.

Result:

$$a_1 = \frac{\alpha}{1-\beta\alpha} \qquad k^{'} = g^{k^{'}}(k) = \beta\alpha z k^{\alpha} \qquad c = g^{c}(k) = (1-\beta\alpha)\, zk^{\alpha}$$

Analytical solution: VFI/Backward induction

$$V^{\mathbf{n+1}}(k) = \max_{\{c,k'\}} \log(c) + \beta V^{\mathbf{n}}\left(k'\right) \qquad \text{s.t. } c + k' = zk^{\alpha}$$

1. Start from initial value, say $V^0(k) = 0$
2. Iterate: $V^1(k) = \max_{k'} \log\left(zk^{\alpha} - k'\right) = \log z + \alpha \log k$
3. Iterate, again: $V^2 = \max_{k'} \log\left(zk^{\alpha} - k'\right) + \beta \log z + \beta\alpha \log k'$
   3.1 Euler: $\frac{1}{zk^{\alpha}-k'} = \frac{\beta\alpha}{k'} \longrightarrow k' = \frac{\beta\alpha}{1+\beta\alpha} zk^{\alpha}$
   3.2 Replace back: $V^2(k) = [\text{Constant}] + (1+\beta\alpha)\,\alpha \log k^{\alpha}$
4. Keep going... you can see that $1 + \beta\alpha + (\beta\alpha)^2 + \ldots = \frac{1}{1-\beta\alpha}$

Result:

$$a_1 = \frac{\alpha}{1-\beta\alpha} \qquad k' = g^{k'}(k) = \beta\alpha z k^{\alpha} \qquad c = g^{c}(k) = (1-\beta\alpha)\, zk^{\alpha}$$

Analytical solution: Euler equation

$$V(k) = \max_{\{c,k'\}} \log(c) + \beta V\left(k'\right) \qquad \text{s.t. } c + k' = zk^{\alpha}$$

Euler equation (obtained with envelope theorem):

$$\frac{1}{zk^{\alpha} - g(k)} = \frac{\beta\alpha z\,(g(k))^{\alpha-1}}{z\,(g(k))^{\alpha} - g(g(k))}$$

Objective is to find the policy function **g** directly

- Guess and verify works here: $g(k) = szk^{\alpha} \longrightarrow s = \beta\alpha$
- More generally we might try to solve this problem numerically
- Fit a parametric function that approximates the solution
- Particularly useful for life cycle models - No need to solve V

Value Function Iteration

Value Function Iteration

Objective is to solve Bellman's equation:

$$V(k,z) = \max_{\{c,k'\}} u(c) + \beta E\left[V\left(k^{'},z^{'}\right)|z\right]$$

$$\text{s.t.} c + k^{'} = f(k,z)$$

$$z^{'} = h(z,\eta); \eta \text{ stochastic}$$

Value Function Iteration

Solution is fixed point of the mapping T:

$$V(k,z) = \mathbf{TV}(\mathbf{k},\mathbf{z}) = \max_{\{c,k'\}} u(c) + \beta E\left[V\left(k^{'},z^{'}\right)|z\right]$$

$$\text{s.t.} c + k^{'} = f(k,z)$$

$$z^{'} = h(z,\eta); \eta \text{ stochastic}$$

Value Function Iteration

CMT gives us a solution by iterating over functions:

$$\mathbf{V^{n+1}}(k,z) = T\mathbf{V^{n}}(k,z) = \max_{\{c,k'\}} u(c) + \beta E\left[\mathbf{V^{n}}\left(k^{'},z^{'}\right)|z\right]$$
$$\text{s.t.} c + k^{'} = f(k,z)$$
$$z^{'} = h(z,\eta)\,;\eta \text{ stochastic}$$

CMT lets us start from an arbitrary function

VFI - Algorithm

Algorithm 1: Value Function Iteration

Result: Fixed Point of Bellman Operator T

$n = 0$; $V^0 \in S$; $dist_V = 1$;
while $n \leq N$ & $dist_V > tol_V$ **do**
  $V^{n+1} = TV^n$;
  $dist_V = d\left(V^{n+1}, V^n\right)$;
end
if $dist_V \leq tol_V$ **then**
  Obtain g from TV^n;
else
  You are in trouble... something went wrong;
end

VFI - Algorithm implementation I

Algorithm 2: VFI: Discrete grid with loops

```
input  : Grid size n_k, model par. z, α, β, code par. max_iter, tol_V
output: Value function V and policy functions G_kp, G_c

k_grid = range(1E-5,2*k_ss;length=n_k) ;
V_old = zeros(n_k) ; iter = 0 ; V_dist = 1 ;

while iter<=max_iter && dist_V>tol_V do
    V_new,G_kp,G_c = T(V_old,k_grid,z, α, β);
    dist_V = maximum(abs.(V_new./V_old.-1)) ;
    iter += 1;

if dist_V<=tol_V then
    return V_new,G_kp,G_c;
else
    error("You are in trouble... something went wrong");
```

VFI - Algorithm implementation II

Algorithm 3: VFI: Discrete grid with loops

input : Grid size n_k, model par. z, α, β, code par. max_iter, tol_V
output: Value function V and policy functions G_kp, G_c

```
k_grid = range(1E-5,2*k_ss;length=n_k) ;
V_old = zeros(n_k) ; iter = 0 ; V_dist = 1 ;
for iter = 1:max_iter do
    V_new,G_kp,G_c = T(V_old,k_grid,z,α,β);
    dist_V = maximum(abs.(V_new./V_old.-1)) ;
    if dist_V<=tol_V then
        return V_new,G_kp,G_c;
error("You are in trouble... max_iter reached");
```

VFI - What does it actually mean?

- It means solving a maximization problem many times
- Inside maximization problem you need expectations

This is hard... and slow... convergence at rate β... but $\beta \approx 1$

- How to speed up?
    1. Speed up solution (EGM)
    2. Skip solution (Howard's PFI)
    3. Speed up update (MPB)

VFI - Grid Search

We will start with the simplest implementation of VFI

- No continuous choice
- Instead choose from a grid (hence grid search)

Why is this useful?

- No derivatives
- Robust to kinks, asymmetries, etc.
- Easy to implement

Limitations

- It is an approximation... not very precise
- Low rate of convergence
- Curse of dimensionality - Pay for precision (and even then)

VFI - Grid Search

Original problem:

$$V\left(k\right) = \max_{\left\{c,k'\right\}} \log\left(c\right) + \beta V\left(k^{'}\right) \qquad \text{s.t. } c + k^{'} = zk^{\alpha}$$

Approximation:

$$V\left(k_i\right) = \max_{k' \in \{k_1,\ldots,k_I\}} \log\left(zk_i^{\alpha} - k^{'}\right) + \beta V\left(k^{'}\right)$$

Note: Everything is a vector or a matrix now

$$\vec{V} = \left[V_1, \ldots, V_I\right]^T \qquad \vec{k} = \left[k_1, \ldots, k_I\right]^T \qquad \vec{U} = \left[U_{ij} = u\left(zk_i^{\alpha} - k_j^{'}\right)\right]$$

VFI - Grid Search - Code I

Algorithm 4: Bellman Operator: Discrete grid with loops

```
Function T(V_old,k_grid,z,α,β):
    n_k = length(k_grid)
    V = zeros(n_k); G_kp = fill(0,n_k); G_c = zeros(n_k)
    for i = 1:n_k do
        V_aux = zeros(n_k)
        for j = 1:n_k do
            V_aux[j] = u(k_grid[i],k_grid[j],z,α,β) + β*V_old[j]
        V[i], G_kp[i] = findmax(V_aux)
        G_c[i] = z*k_grid[i]^α - k_grid[G_kp[i]]
    return V, G_kp, G_c
```

VFI - Grid Search - Code II

Algorithm 5: Bellman Operator: Discrete grid with matrices

Function `T`($V\_old, U\_mat, k\_grid, z, \alpha, \beta$):

n_k = length(V_old)

V, G_kp = findmax(U_mat .+ β*repeat(V_old',n_k,1) , dims=2)

G_kp = [G_kp[i][2] for i in 1:n_k]

G_c[i] = z*k_grid[i]^α - k_grid[G_kp[i]]

return V, G_kp, G_c

Where:

U_mat = [utility(k_grid[i],k_grid[j],z, α, β) for i in 1:n_k, j in 1:n_k]

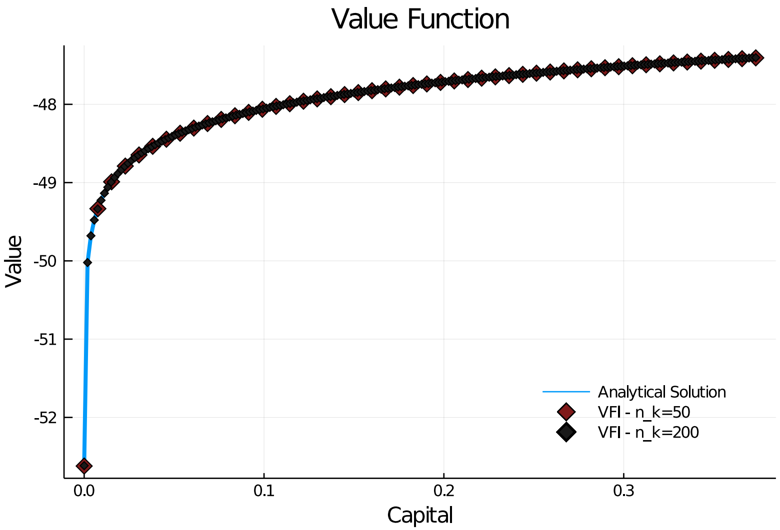
How do we judge the solution?

- Plot as much as you can
- Summary statistics can hide large mistakes
- Report what is most relevant for what you are doing

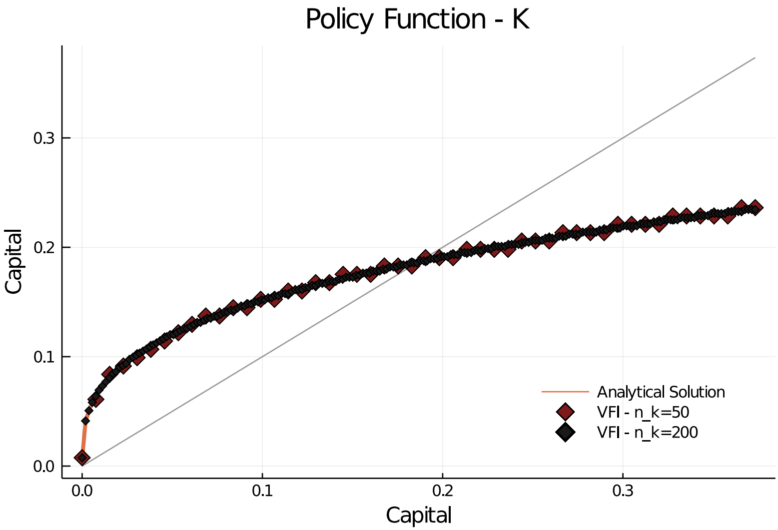
In this case we know the solution

1. Plot value function
2. Plot policy function

Value and policy functions



Value and policy functions



Judging the solution

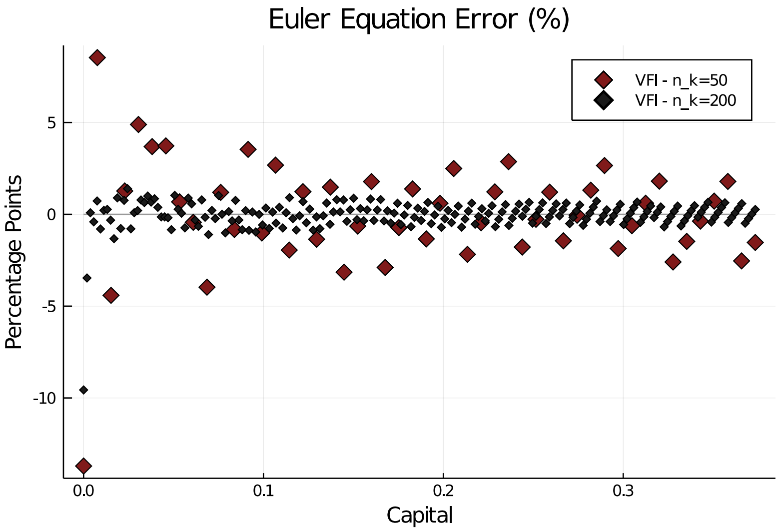
- Graphs point at a great fit
  - Even with $n_k = 50$ the fit is really good
  - $n_k = 200$ seems more than enough
- But these graphs can be misleading
  - They are approximations: Discrete problem vs continuous problem

Judge the solution with the optimization of the agent:

$$\frac{1}{zk^{\alpha} - g(k)} = \frac{\beta\alpha z (g(k))^{\alpha-1}}{z(g(k))^{\alpha} - g(g(k))}$$

$$0 = \underbrace{\frac{\beta\alpha z (g(k))^{\alpha-1} zk^{\alpha} - g(k)}{z(g(k))^{\alpha} - g(g(k))} - 1}_{\text{\% Error in Euler Equation}}$$

Euler Equation - Not a great fit



Howard's Policy Iteration

Howard's policy iteration: The idea

- The hardest step for VFI is the maximization step
  - Even for discrete grid

  Using the policy function only once is such a waste...

- Howard's policy iteration:

  Solve for the policy function once and use it to update many times!

$$V^{n+1}(k) = T^H V^n = u(\overline{\mathbf{c}}(\mathbf{k})) + \beta V^n\left(\overline{\mathbf{k}}^{\prime}(\mathbf{k})\right)$$

where $\overline{c}$ and $\overline{k}^{\prime}$ are fixed policy functions

Howard's policy iteration: The idea

Why would applying the same policy function many times work?

- Turns out the mapping T^H with given $\overline{c}$ and $\overline{k}'$ is also a contraction.
- So the iteration process will converge to a unique fixed point...
  just not to the solution to our original problem

So, why do policy iteration?

- Algorithm does not necessarily take us where we want, but it (can) take us close and fast (mostly fast)

Howard's policy iteration

Algorithm 6: VFI with Howard's Policy Iteration

Result: Fixed Point of Bellman Operator T

$n = 0$; $V^0 \in S$; $dist_V = 1$;

while $n \leq N$ & $dist_V > tol_V$ **do**

% Compute current policy function ;

$G^n = \text{argmax}\{TV^n\}$;

% Obtain fixed point under G^n ;

$V^{n+1} = \lim_{m\to\infty} T^m_{G^n} V^n$;

$dist_V = d\left(V^{n+1}, V^n\right)$;

end

Howard's policy iteration: Properties

Results from Puterman & Brumelle (1979)

- Policy iteration is equivalent to the Newton-Kantorovich method in the context of dynamic programming
- HPI behaves like Newton's method:
    1. The method is guaranteed to converge if initial guess is in some neighborhood of the true solution ("Basin of Attraction").
    2. If $V_0 \in$"Basin of Attraction" the method converges at a quadratic rate in the iteration index n.

Howard's policy iteration

- So the new algorithm is potentially very fast ...

  But it no longer has the global convergence properties of VFI
- Quadratic convergence is misleading because it operates over n
  - Each iteration takes a long time because we want the fixed point of T_G
- Overall it is not clear that it is faster...

  To make matters worse the "Basin of Attraction" can be small (and is definitely unknown)

Solution: Use the policy iteration only for n_H steps

(Modified) Howard's policy iteration

Algorithm 7: VFI with Howard's Policy Iteration

Result: Fixed Point of Bellman Operator T

$n = 0$; $V^0 \in S$; $dist_V = 1$;

while $n \leq N$ & $dist_V > tol_V$ **do**

  % Compute current policy function ;

    $G^n = \text{argmax}\{TV^n\}$;

  % Iterate n_H times under G^n ;

    $V^{n+1} = T_{G^n}^{n_H} V^n$;

  $dist_V = d\left(V^{n+1}, V^n\right)$;

end

HPI: Algorithm Implementation

Algorithm 8: Howard's Policy Iteration

Function T^{HPI}(*V_old,U_mat,k_grid,z*, α, β,***n_H***):

```
n_k = length(V_old)
V, G_kp = findmax( U_mat .+ β*repeat(V_old',n_k,1) , dims=2)
U_vec = U_mat[G_kp]
for i=1:n_H do
    V = U_vec .+ b*repeat(V_old',n_k,1)[G_kp]
    if maximum(abs.(V./V_old.-1))<=tol then
        break
    V_old = V
G_kp = [G_kp[i][2] for i in 1:n_k]
G_c[i] = z*k_grid[i]^α - k_grid[G_kp[i]]
return V, G_kp, G_c
```

MacQueen-Porteus Bounds

Convergence and Stopping Criteria

How do we know when we are close to the solution?

- The CMT gives us an answer for VFI:

$$d\left(V^{\star}, V^{n}\right) \leq \frac{1}{1-\beta} d\left(V^{n}, V^{n-1}\right)$$

- Stop if ϵ away from solution: $d\left(V^{n}, V^{n-1}\right) \leq \epsilon\left(1-\beta\right)$

This bound on distance is not too informative:

- Bound is a worst case scenario (and covers all the function's domain)

MacQueen-Porteus Bounds

Can we get a better bound for how far we are from the solution?

- The MacQueen-Porteus Bounds (MPB) provide us with better bounds
  - New bounds close faster, they are more informative
  - But for a different specification of the DP problem

Discrete-State Dynamic Programming:

$$V(x_i) = \max_{y \in \Gamma(x_i)} \left\{ U(x_i, y) + \beta \sum_{j=1}^{N_x} \pi_{ij}(y) V(x_j) \right\}$$

- State x is discrete but control y is continuous
- Transition matrix depends on control: $\Pi(y)$
- Very common in other fields
  - See Bertsekas & Shreve (1996) or Bertsekas & Ozdaglar (2003)

MacQueen-Porteus Bounds

Theorem

Consider the discrete-state dynamic programming problem

$$V^n(x_i) = TV^{n-1}(x_i) = \max_{y \in \Gamma(x_i)} \left\{ U(x_i, y) + \beta \sum_{j=1}^{N_x} \pi_{ij}(y) V^{n-1}(x_j) \right\}$$

Define $\underline{c}_n = \frac{\beta}{1-\beta} \min\{V_n - V_{n-1}\} \quad \wedge \quad \overline{c}^n = \frac{\beta}{1-\beta} \max\{V_n - V_{n-1}\}$
Then, for all $x \in X$ *and* V^0*, it holds that:*

$$T^n V^0(x) + \underline{c}_n \leq V^\star(x) \leq T^n V^0(x) + \overline{c}_n$$

Further, the two bounds approach the solution monotonically as n grows.

MacQueen-Porteus Bounds - Algorithm

Algorithm 9: VFI with MacQueen-Porteus Bounds

Result: Fixed Point of Bellman Operator T

$n = 1$; $V^0 \in S$; $dist_V = 1$;

while $n \leq N$ & $dist_V > tol_V$ **do**

$\quad V^n = TV^n - 1$;

$\quad \underline{c}_n = \frac{\beta}{1-\beta} \min\left\{V^{n+1} - V^n\right\}$; $\qquad \bar{c}_n = \frac{\beta}{1-\beta} \max\left\{V^{n+1} - V^n\right\}$;

$\quad dist_V = \bar{c}_n - \underline{c}_n$;

end

$V = V^n + \frac{\bar{c}_n + \underline{c}_n}{2}$;

$G = \operatorname{argmax} TV$;

MacQueen-Porteus Bounds - Properties

Results from Bertsekas (1987)

- The MPB converge monotonically to the true solution
- Convergence is proportional to the subdominant eigenvalue of $\Pi(y^\star)$ (transition matrix evaluated at the optimal policy)
  - For an AR(1) process the subdominant eigenvalue is ρ (persistence)
  - If persistence is low convergence is very fast
- Compare with VFI:
  - Convergence proportional to dominant eigenvalue
  - Always 1 because Π is a stochastic matrix
  - Multiplied by β gives convergence rate... but we often have $\beta \approx 1$

Coda: Convergence in policy functions

- What does it mean to be ϵ away for the value function?
  - Hard to interpret the level of the value function
- For most applications the level of the policy functions is more relevant
  - It is clearly more interpretable: ϵ% of consumption or capital
- Comparing policy functions is more efficient
  - Policy functions also converge faster than value functions
  - Reduce computation time
- Value functions critical for welfare comparisons