Value Function Methods

Quantitative Macroeconomics [Econ 5725]

Raül Santaeulàlia-Llopis

Washington University in St. Louis

Spring 2016

Value Function Methods

The value function iteration algorithm (VFI) described in our previous set of slides [Dynamic Programming.pdf] is used here to solve for the value function in the neoclassical growth model. We will discuss first the deterministic model, then add a productivity shock to discuss the stochastic version. [1]

We introduce two types of value function methods:

- First, we describe global methods to approximate for the value function using discretization, finite-element methods, and weighted residual methods.
- Second, we look into local methods to solve for the value function using linear-quadratic approximations.

[1] These slides build on notes by Carlos Urrutia 'Metodos Numéricos para Resolver Modelos Macroeconómicos Dinámicos', and the courses by José-Víctor Ríos-Rull and Makoto Nakajima (check their respective websites). Read further textbook references in the syllabus.

The Model

Sequential formulation. Consider a decentralized economy populated by a large number, N_t, of identical infinitely lived households. Each of them maximizes the discounted future stream of utility,

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} E_0 \sum_{t=0}^{\infty} \beta^t u(c_t, 1),$$

subject to a budget constraint,

$$c_t + k_{t+1} \quad = \quad w_t + (1 + r_t - \delta)k_t,$$

given k_0 and z_0.

An aggregate firm maximizes profits (an static problem),

$$Y_t - w_t N_t - r_t K_t,$$

subject to aggregate technology

$$Y_t = F(K_t, N_t).$$

Remark. In Homework 3, you were asked to solve for the sequential formulation of the neoclassical model (solving directly for the allocations, i.e., the time-path of capital) when asked to compute a transition between steady states. Here, we are going to learn how to solve for the recursive formulation of the neoclassical model.

Remark. For an equivalence theorem between these two formulations see SLP.

Recursive Competitive Equilibrium (RCE)

A RCE is a set of functions $v(k, K)$, $c(k, K)$, $k'(k, K)$; prices $w(K)$ and $r(K)$; and an aggregate law of motion $\Gamma(K)$ such that,

1. Given w, r and Γ, the value function $v(k, K)$ solves the Bellman equation,

$$v(k, K) = \max_{c \geq 0, k'} u(c) + \beta v(k', K') \tag{1}$$

such that

$$\begin{aligned} c + i &= w(K) + r(K)k \\ k' &= i + (1 - \delta)k \\ K' &= \Gamma(K). \end{aligned}$$

Associated to the value function we have the set of optimal decision rules $c(k, K)$ and $k'(k, K)$.

2. Factor prices equate marginal products, $r(K) = f'(K)$ and $w(K) = f(K) - f'(K)K$.
3. Market clearing, $f(K) = c(k, K) + i(k, K)$.
4. Aggregate consistency, $\Gamma(K) = k'(K, K)$.

Pareto Optimal Equilibrium

Remark. Welfare theorems hold, that is, the RCE is Pareto Optimal. We define the recursive PO equilibrium as a set of functions $v(k)$, $c(k)$, $k'(k)$ that solve

$$v(k) = \max_{c \geq 0, k'} u(c) + \beta v(k') \tag{2}$$

such that

$$\begin{aligned} c + i &= f(k) \\ k' &= i + (1 - \delta)k \end{aligned}$$

That is, the PO problem is

$$v(k) = \max_{k' \in [0, f(k) + (1-\delta)k]} u(f(k) + (1 - \delta)k - k') + \beta v(k') \tag{3}$$

Remark. Remember that the Bellman principal of optimality ensures that we can recover the sequential solution using the recursive solution. To see this, note that starting with an initial k_0 we can use the optimal decision rules of the recursive problem to, recursively, get the sequence

$$\begin{aligned} k_1 &= k'(k_0) \\ k_2 &= k'(k_1) = k'(k'(k_0)) \end{aligned}$$

and so on for all $t \geq 3$.

VFI with Discretization to Approximate V

Remark. Remember that the value function algorithm (VFI) holds independently of what method we use to approximate V.

Here, we will start by discretezicing V to approximate the value function. Later we will approximate V with finite element methods, as well as weighted residual methods that we have seen in our previous sessions.

- **STEP 1**: The continuous variable k is discretized by defining a grid for the values of k, that is, $K = (k_1, k_2, ..., k_i, ..., k_p)$ with $k_1 = k_{min}$, and $k_p = k_{max}$. For example, an evenly spaced grid is

$$k_i = k_{min} + (i - 1)\,\eta$$

with $\eta = \frac{k_{max} - k_{min}}{p-1}$. Obviously, with higher p, and smaller η, the approximation will be more precise, but more costly to compute.

For the k_{min} we should choose a value slightly higher than zero to avoid potential problems of having zero output, and perhaps zero consumption. For k_{max} there is no rule, but its choice may depend on what we are interested in. For instance, if we know the steady state k^*, and we are interested only in dynamics below the steady state, we can safely choose k_{max} to be slightly above the steady state.

Here we choose the discretatization to uniformily distributed over the state space. However, if we knew where a potential kink is, or the regions of higher curvature, we might be interested in clustering points in those regions. That is, we would not need much points in regions where the value function is known to be linear (or close to).

- **STEP 2**: Guess a solution $V^{s=0} \in R^p$, say the null vector, that is, $V^{s=0}(k_i) = 0$ for all $i = \{1, ..., p\}$. s keeps track of the number of iterations.

  An alternative is to set $V_i^{s=0} = \frac{u(f(k_i)+(1-\delta)k_i-k_i)}{1-\beta}$. That is, $k_j = k_i$.

- **STEP 3**: Define the return matrix M as follows,

$$M = \begin{bmatrix} m(k_1, k_1) & m(k_1, k_2) & ... & m(k_1, k_p) \\ m(k_2, k_1) & m(k_2, k_2) & ... & m(k_2, k_p) \\ ... & ... & ... & ... \\ m(k_p, k_1) & m(k_p, k_2) & ... & m(k_p, k_p) \end{bmatrix}.$$

The generic element in matrix M contains the return funcion

$$M_{i,j} = m(k_i, k_j) = u\left(f(k_i) + (1 - \delta)k_i - k_j\right)$$

evaluated at all possible combinations $k_i \in K$ and $k_j \in K$.

- **STEP 4**: However, we know that not all values for c, hence k', are feasible. The nonnegativity restriction on consumption $c_{ij} = f(k_i) + (1-\delta)k_i - k_j \geq 0$, implies that $k_j \leq f(k_i) + (1-\delta)k_i$.

  Therefore, we replace $M_{ij} = m(k_i, k_j) = \omega$ if $k_j > f(k_i) + (1-\delta)k_i$, where ω is a very negative number.

- **STEP 5**: Given the vector $V^s = (V_1^s, V_2^s, ..., V_i^s, ..., V_p^s)$ and the matrix M:
  - **STEP 5.1**: Compute the matrix, χ,

$$\chi = \begin{bmatrix} M_{11} + \beta V_1^s & M_{12} + \beta V_2^s & ... & M_{1p} + \beta V_p^s \\ M_{21} + \beta V_1^s & M_{22} + \beta V_2^s & ... & M_{2p} + \beta V_p^s \\ ... & ... & ... & ... \\ M_{p1} + \beta V_1^s & M_{p2} + \beta V_2^s & ... & M_{pp} + \beta V_p^s \end{bmatrix}.$$

    where the generic element of χ is

$$\chi_{i,j} = M_{i,j} + \beta V_j^s.$$

  - **STEP 5.2**: Compute the updated value function V^{s+1} as the maximum element in each row of χ,

$$V^{s+1} = \begin{bmatrix} \max\{\chi_{11}, \chi_{12}, ..., \chi_{1p}\} \\ \max\{\chi_{21}, \chi_{22}, ..., \chi_{2p}\} \\ ..... \\ \max\{\chi_{p1}, \chi_{p2}, ..., \chi_{pp}\} \end{bmatrix}.$$

- **STEP 6**: If $||V^{s+1} - V^s|| < \varepsilon$, stop and report success. Otherwise, go back to the previous step, **STEP 5**, replacing $s = s + 1$.

- **STEP 7**: Being thorough.

    - Check the bounds of the state space are not binding. Otherwise, relax them.

    - Make sure ε is small enough. Reduce ε, redo the VFI, and check your answer does not change. Otherwise, keep reducing ε.

    - Make sure the size of the grid, p, is large enough. Increase p, redo the VFI, and check your answer does not change. Otherwise, keep increasing p.

- **Remark**. Once we know V^s, the decision rule is obtained as

$$g^s = \begin{bmatrix} \arg\max\{\chi_{11}, \chi_{12}, ..., \chi_{1p}\} \\ \arg\max\{\chi_{21}, \chi_{22}, ..., \chi_{2p}\} \\ ..... \\ \arg\max\{\chi_{p1}, \chi_{p2}, ..., \chi_{pp}\} \end{bmatrix}.$$

  That is, g^s is a column vector attaining values $g_i^s \in \{1, ..., p\}$ that state the column number j that maximizes row i.

- **Remark**. It is going to be useful to store the decision rule at each iteration s for the purposes of implementing some of the speeding-up algorithms that we will see next.

- **Remark**. We can then find the optimal sequence of capital for a given initial capital.

Speeding-up the VFI Algorithm

- We explore four ways of speeding up the VFI algorithm,
    - Using the monotonicity of the optimal decision rule.
    - Using the concavity of the value function.
    - Local search.
    - Howard's policy iteration step.

Speeding-up the VFI Algorithm I: Monotonicity of the optimal decision rule

- Here we pursue to reduce the number of points to be searched on the grid by the VFI algorithm.
- To do so, we use the known property that the optimal decision rule increases in K. Therefore, if $k_j > k_i$, then $g(k_j) \geq g(k_i)$.[2]
- Hence, if we want to find $g(k_j)$ when $k_j > k_i$, we can rule out searching for all grid values of capital that are smaller than $g(k_i)$.
- This means we can modify **STEP 5.1** in the VFI.

[2] When strictness applies the equality arises if grids are too loose.

- **Replace STEP 5.1**

  - For each k_i, set an optimal lower bound for $g^s(k_i)$ as

    $$\underline{g^s(k_i)} = g^s(k_{i-1}).$$

  - Compute the matrix χ only for the pairs (k_i, k_j) that satisfy the lower bound, $k_j \geq \underline{g^s(k_i)}$.

  - That is, not all elements in χ need to be computed.

  - Note that you will have to store the decision rule at the end of each iteration to implement this improvement.

Speeding-up the VFI Algorithm II: Concavity of the value function

- We use the known property that the maximand in the Bellman equation, $M_{i,j} + \beta V_j$, is strictly concave in k'.
- Therefore, if $M_{i,j} + \beta V_j > M_{i,j+1} + \beta V_{j+1}$, then $M_{i,j} + \beta V_j > M_{i,j+2} + \beta V_{j+2}$.
- This means we can modify **STEP 5.1** in the VFI to improve the efficiency of the algorithm.

- **Replace STEP 5.1**

  - When computing the elements of the matrix χ apply the following loop to each row i,

    do $j = \{1, p\}$

    $$\chi_{i,j} = M_{i,j} + \beta V_j$$

    if $\chi_{i,j-1} > \chi_{i,j}$, stop & report the optimal value is $\chi_{i,j-1}$.
    end do

  - That is, not all elements in χ need to be computed.

Speeding-up the VFI Algorithm III: Local search

- We exploit the property that the optimal decision rule is continuous.
- Hence, if we know that $k_j = g(k_i)$, and we we have reasonable fine grids, then we hope that $g(k_{i+1})$ is in a small neigborhood of k_j. This restricts the elements of χ that need to be computed.
- Local search is followed by practioners as it has been proven useful. But there is no theorem behind this.
- We can make sure that our solution is the one that is not limited to search locally by checking that the bounds that define the neigborhood are not binding at each iteration.
- Note that you will have to store the decision rule at the end of each iteration to implement this improvement.

Speeding-up the VFI Algorithm IV: Howard's policy iteration

- This method does not rely on the properties of the optimal value function, or decision rule.
- We can apply a guessed decision rule many times to update the value function many times, without solving for the optimal rule at each iteration.
- We will need to decide how many times, n_h, to update the value function without updating the decision rule.
- If we have a decision rule that is not far from the optimal one, higher n_h implies faster convergence, but iterating on the value function while keeping a decision rule obtained in early stages of the VFI might not be that useful.
- Note that you will have to store the decision rule at the end of each iteration to implement this improvement.

Remark. Note that the speeding-up algorithms are not exclusive. We can use any combination of them to speed up the VFI.

The Stochastic Model

- Productivity is subject to a stochastic process. At each period, output is

$$Y_t = z_t f(K_t)$$

and decisions are taken before the realization of z_t is observed.

- To apply the VFI we assume that z_t takes a finite number of q values, $Z \in (z_1, z_2, ..., z_q)$, and follows a first order Markov process that has a square transition matrix $\Pi_{q \times q}$ with generic element

$$\pi_{l,l'} = Prob[z_{t+1} = z_{l'} | z_t = z_l],$$

with $\sum_{l'=1}^{q} \pi_{l,l'} = 1$.

A stochastic RCE is a set of functions $v(k, K, z)$, $c(k, K, z)$, $k'(k, K, z)$; prices $w(K, z)$ and $r(K, z)$; and an aggregate law of motion $\Gamma(K, z)$ such that,

1. Given w, r and Γ, the value function $v(k, K, Z)$ solves the Bellman equation,

$$v(k, K, z) = \max_{c \geq 0, k'} u(c) + \beta E_Z v(k', K', z') = \max_{c \geq 0, k'} u(c) + \beta \sum_{l'=1}^{q} \pi_{l,l'} v(k', K', z')$$

   such that

$$\begin{aligned} c + i &= w(K, z) + r(K, z)k \\ k' &= i + (1 - \delta)k \\ K' &= \Gamma(K, z) \end{aligned}$$

   Associated to the value function we have the set of optimal decision rules $c(k, K, z)$ and $k'(k, K, z)$.

2. Factor prices equate marginal products, $r(K, z) = zf'(K)$ and $w(K, z) = zf(K) - zf'(K)K$.
3. Market clearing, $zf(K) = c(k, K, z) + i(k, K, z)$.
4. Aggregate consistency, $\Gamma(K, z) = k'(K, K, z)$.

- **Remark**. To obtain the optimal time paths for we need to simulate a path for the productivity shocks.

- **Remark**. Welfare theorems hold. The stochastic recursive PO equilibrium as a set of functions $v(k,z)$, $c(k,z)$, $k'(k,z)$ that solve

$$v(k,z) = \max_{c\geq 0, k'} u(c) + \beta E_Z v(k',z') = \max_{c\geq 0, k'} u(c) + \beta \sum_{l'=1}^{q} \pi_{l,l'} v(k',z') \qquad (4)$$

such that

$$\begin{aligned} c + i &= zf(k) \\ k' &= i + (1-\delta)k \end{aligned}$$

That is, the PO problem is

$$v(k) = \max_{k'\in[0, zf(k)+(1-\delta)k]} u(zf(k) + (1-\delta)k - k') + \beta \sum_{l'=1}^{q} \pi_{l,l'} v(k',z') \qquad (5)$$

VFI with discretization to approximate V in a stochastic model

- **STEP 1**: Define a grid for the values of capital k_i, that is, $K = (k_1, k_2, ..., k_i, ..., k_p)$ and a grid for the values of the productivity shock z_i, $Z = (z_1, z_2, ..., z_q)$.

- **STEP 2**: Define the matrix M as follows,

$$M_{pq\times p} = \begin{bmatrix} m(k_1,k_1,z_1) & m(k_1,k_2,z_1) & \dots & m(k_1,k_p,z_1) \\ m(k_2,k_1,z_1) & m(k_2,k_2,z_1) & \dots & m(k_2,k_p,z_1) \\ \dots & \dots & \dots & \dots \\ m(k_p,k_1,z_1) & m(k_p,k_2,z_1) & \dots & m(k_p,k_p,z_1) \\ m(k_1,k_1,z_2) & m(k_1,k_2,z_2) & \dots & m(k_1,k_p,z_2) \\ m(k_2,k_1,z_2) & m(k_2,k_2,z_2) & \dots & m(k_2,k_p,z_2) \\ \dots & \dots & \dots & \dots \\ m(k_p,k_1,z_2) & m(k_p,k_2,z_2) & \dots & m(k_p,k_p,z_2) \\ \dots & \dots & \dots & \dots \\ m(k_1,k_1,z_q) & m(k_1,k_2,z_q) & \dots & m(k_1,k_p,z_q) \\ m(k_2,k_1,z_q) & m(k_2,k_2,z_q) & \dots & m(k_2,k_p,z_q) \\ \dots & \dots & \dots & \dots \\ m(k_p,k_1,z_q) & m(k_p,k_2,z_q) & \dots & m(k_p,k_p,z_q) \end{bmatrix}.$$

The generic element in matrix M contains the return function

$$M_{p(l-1)+i,j} = m(k_i,k_j,z_l) = u\left(z_l f(k_i) + (1-\delta)k_i - k_j\right)$$

evaluated at all possible combinations $k_i \in K$, $k_j \in K$, and $z_l \in Z$.

- **STEP 3**: However, we know that not all values for c, hence k', are feasible. The nonnegativity restriction on consumption implies,

$$M_{p(l-1)+i,j} = m(k_i, k_j, z_l) = \omega$$

if $k_j > z_l f(k_i) + (1-\delta)k_i$, where ω is a very negative number.

- **STEP 4**: Guess a solution $V^{s=0} \in R^{pq}$, say the null vector, that is, $V^{s=0}(k_i, z_l) = 0$ for all $i = \{1, ..., p\}$ and $l = \{1, ..., q\}$. s keeps track of the number of iterations.

- **STEP 5**: Given the vector $V^s = (V_1^s, V_2^s, ..., V_{pq}^s)$, with generic element

$$V_{p(l'-1)+j}^s = V^s(k_j, z_{l'}),$$

compute the following in-steps:

- **Step 5.0**: Compute the matrix $W_{q\times p}^s$,

$$\begin{bmatrix} \sum_{l'=1}^q \pi_{1,l'} V_{p(l'-1)+1}^s & \sum_{l'=1}^q \pi_{1,l'} V_{p(l'-1)+2}^s & \cdots & \sum_{l'=1}^q \pi_{1,l'} V_{p(l'-1)+p}^s \\ \sum_{l'=1}^q \pi_{2,l'} V_{p(l'-1)+1}^s & \sum_{l'=1}^q \pi_{2,l'} V_{p(l'-1)+2}^s & \cdots & \sum_{l'=1}^q \pi_{2,l'} V_{p(l'-1)+p}^s \\ \cdots & \cdots & \cdots & \cdots \\ \sum_{l'=1}^q \pi_{q,l'} V_{p(l'-1)+1}^s & \sum_{l'=1}^q \pi_{q,l'} V_{p(l'-1)+2}^s & \cdots & \sum_{l'=1}^q \pi_{q,l'} V_{p(l'-1)+p}^s \end{bmatrix},$$

with generic element

$$W_{l,j}^s = \sum_{l'=1}^q \pi_{l,l'} V_{p(l'-1)+j}^s$$

- ... (continued)
  - **Step 5.1**: Compute the matrix $\chi_{pq\times p}$,

$$\begin{bmatrix}
M_{11}+\beta W^s_{11}, & M_{12}+\beta W^s_{12}, & ..., & M_{1p}+\beta W^s_{1p} \\
M_{21}+\beta W^s_{11}, & M_{22}+\beta W^s_{12}, & ..., & M_{2p}+\beta W^s_{1p} \\
... & ... & ... & ... \\
M_{p1}+\beta W^s_{11}, & M_{p2}+\beta W^s_{12}, & ..., & M_{pp}+\beta W^s_{1p} \\
M_{p+1,1}+\beta W^s_{21}, & M_{p+1,2}+\beta W^s_{22}, & ..., & M_{p+1,p}+\beta W^s_{2p} \\
M_{p+2,1}+\beta W^s_{21}, & M_{p+s,2}+\beta W^s_{22}, & ..., & M_{p+2,p}+\beta W^s_{2p} \\
... & ... & ... & ... \\
M_{p+p,1}+\beta W^s_{21}, & M_{p+p,2}+\beta W^s_{22}, & ..., & M_{p+p,p}+\beta W^s_{2p} \\
... & ... & ... & ... \\
M_{p(q-1)+1,1}+\beta W^s_{q1}, & M_{p(q-1)+1,2}+\beta W^s_{q2}, & ..., & M_{p(q-1)+1,p}+\beta W^s_{qp} \\
M_{p(q-1)+2,1}+\beta W^s_{q1}, & M_{p(q-1)+2,2}+\beta W^s_{q2}, & ..., & M_{p(q-1)+2,p}+\beta W^s_{qp} \\
... & ... & ... & ... \\
M_{p(q-1)+p,1}+\beta W^s_{q1}, & M_{p(q-1)+p,2}+\beta W^s_{q2}, & ..., & M_{p(q-1)+p,p}+\beta W^s_{qp}
\end{bmatrix},$$

with the generic element

$$\chi_{p(l-1)+i,j} = M_{p(l-1)+i,j} + \beta W^s_{l,j}$$

- ... (continued)
  - **Step 5.2**: Compute the updated value function V^{s+1} as the maximum element in each row of χ,

$$V^{s+1} = \begin{bmatrix} \max\{\chi_{1,1}, & \chi_{1,2}, & \ldots, & \chi_{1,p}\} \\ \max\{\chi_{2,1}, & \chi_{2,2}, & \ldots, & \chi_{2,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \max\{\chi_{p,1}, & \chi_{p,2}, & \ldots, & \chi_{p,p}\} \\ \max\{\chi_{p+1,1}, & \chi_{p+1,2}, & \ldots, & \chi_{p+1,p}\} \\ \max\{\chi_{p+2,1}, & \chi_{p+2,2}, & \ldots, & \chi_{p+2,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \max\{\chi_{p+p,1}, & \chi_{p+p,2}, & \ldots, & \chi_{p+p,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \ldots, & \ldots, & \ldots, & \ldots \\ \max\{\chi_{p(q-1)+1,1}, & \chi_{p(q-1)+1,2}, & \ldots, & \chi_{p(q-1)+1,p}\} \\ \max\{\chi_{p(q-1)+2,1}, & \chi_{p(q-1)+2,2}, & \ldots, & \chi_{p(q-1)+2,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \max\{\chi_{p(q-1)+p,1}, & \chi_{p(q-1)+p,2}, & \ldots, & \chi_{p(q-1)+p,p}\} \end{bmatrix}.$$

- **STEP 6**: If $||V^{s+1} - V^s|| < \epsilon$, stop and report success. Otherwise, go back to the previous step, **STEP 5**, replacing $s = s + 1$.

- **Remark**. We can store the decision rule as

$$g^s = \begin{bmatrix} \arg\max\{\chi_{1,1}, & \chi_{1,2}, & \ldots, & \chi_{1,p}\} \\ \arg\max\{\chi_{2,1}, & \chi_{2,2}, & \ldots, & \chi_{2,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \arg\max\{\chi_{p,1}, & \chi_{p,2}, & \ldots, & \chi_{p,p}\} \\ \arg\max\{\chi_{p+1,1}, & \chi_{p+1,2}, & \ldots, & \chi_{p+1,p}\} \\ \arg\max\{\chi_{p+2,1}, & \chi_{p+2,2}, & \ldots, & \chi_{p+2,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \arg\max\{\chi_{p+p,1}, & \chi_{p+p,2}, & \ldots, & \chi_{p+p,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \ldots, & \ldots, & \ldots, & \ldots \\ \arg\max\{\chi_{p(q-1)+1,1}, & \chi_{p(q-1)+1,2}, & \ldots, & \chi_{p(q-1)+1,p}\} \\ \arg\max\{\chi_{p(q-1)+2,1}, & \chi_{p(q-1)+2,2}, & \ldots, & \chi_{p(q-1)+2,p}\} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \arg\max\{\chi_{p(q-1)+p,1}, & \chi_{p(q-1)+p,2}, & \ldots, & \chi_{p(q-1)+p,p}\} \end{bmatrix}.$$

- Explicitly, the optimal value function $V(k, z)$ and the decision rule $g(k, z)$ take the form of vectors,

$$V = \begin{bmatrix} v(k_1, z_1) \\ v(k_2, z_1) \\ \dots \\ v(k_p, z_1) \\ v(k_1, z_2) \\ v(k_2, z_2) \\ \dots \\ v(k_p, z_2) \\ \dots \\ v(k_1, z_q) \\ v(k_2, z_q) \\ \dots \\ v(k_p, z_q) \end{bmatrix} \qquad g = \begin{bmatrix} g(k_1, z_1) \\ g(k_2, z_1) \\ \dots \\ g(k_p, z_1) \\ g(k_1, z_2) \\ g(k_2, z_2) \\ \dots \\ g(k_p, z_2) \\ \dots \\ g(k_1, z_q) \\ g(k_2, z_q) \\ \dots \\ g(k_p, z_q) \end{bmatrix}$$

- To find the optimal sequence $k_0, k_1, ..., k_T$ first we need to simulate a history of realizations of productivity shocks $z_0, z_1, ..., z_T$. To simulate these productivity shocks we use the transition matrix Π.

- Using the simulated series $z_0, z_1, ..., z_T$, and an initial k_0, we can find the optimal sequence $k_0, k_1, ..., k_T$ using the decision rule g.

- Given this time-series for the productivity shock and capital, we can recover the all time-series in the model, consumption, output and factor prices.

- Therefore, we can compute statistics on variance, correlations, and joint dynamics. Before doing so, we usually filter them using some of the techniques in our previous slides [Data: Some Tools and Sources.pdf].

Solving the competitive equilibrium directly using VFI

- VFI is fairly standard as a solution method for pareto optimal economies, but not so commonly used for decentralized economies.
- The fact that we have two states, k and K, is not the reason. We have seen how to deal with two states earlier in the economy with stochastic productivity shocks.
- The problem of using VFI in decentralized economies is that agents need to know (at least, in expected terms) the aggregate law of motion (i.e., future prices) before optimizing, while out of the VFI we only know this object after solving for the equilibrium.
- One way to circumvent that problem is to add one STEP to the VFI algorithm. What this STEP does is provide a guess a for the aggregate law of motion (i.e., this is again a function approximation problem). Given that guess for the aggregate law of motion, solve for the decentralized economy using VFI. Once the optimal decision rule is obtained, we compare it to the aggregate law of motion. If they are close, we stop. Otherwise we update the aggregate law of motion to be the last optimal decision rule and start again the VFI.
- There is no guarantee of convergence.

Some limitations

- While highly applied to economies in which the welfare theorem holds, it is not that easy to solve directly for the competitive equilibrium using value function iteration.
- Increasing the state variables (or the number of points in the grid) implies the dimension of the matrix χ will grow exponentially making the algorithm very time-consuming.
- We can only introduce shocks of a particular structure, primarily, Markov processes of order one.

VFI with continuous methods to approximate V

- Here we use a continuous method to approximate for the value function within the value function algorithm that solves for the value function.

- We approximate the value function using a linear combination of a set of m known linearly independent basis functions $\psi_j(k)$, $j = 0, ..., m$

$$\widetilde{V}(k) = \sum_{j=0}^{m} \theta_j \psi_j(k) \tag{6}$$

  with m-basis coefficients θ_j to be determined.

- That is, solving for $V(k)$ reduces to find the θ_j coefficients.

- In our working example, the continuous method that we use to approximate for $\widetilde{V}(k)$ is finite elements. That is, we are going to choose the basis functions to be nonzero over some subintervals of the state space, but not all. [3]

- Precisely, we use the B-spline formulation to approximate $\widetilde{V}(k)$. This formulation subscribes to (6). [4]

- What follows applies to B-splines of any order.

[3] Alternatively, we could have chosen the basis functions $\psi_j(k)$ to defined over the whole state space (i.e., spectral approximation methods). For example, $\psi_j(k)$ could be monomial basis, or Chebyshev basis.

[4] Recall from our previous session [Function Approximation], Theorem: every spline function of a given degree, smoothness, and domain paritition, can be represented as a linear combination of B-splines of that same degree and smoothness, and over that partition (de Boor 1978).

- For instance, an order-2 spline implements tent function interpolation (piecewise linear), and are spanned by the B^1-splines.

  That is, the typical B^1-spline is defined by (6) with the following basis functions

$$\psi_j(k) = \begin{cases} \frac{k-k_{j-1}}{k_j-k_{j-1}}, & \text{if} \quad k_{j-1} \leq k \leq k_j \\ \frac{k_{j+1}-k}{k_{j+1}-k_j}, & \text{if} \quad k_j < k \leq k_{j+1} \\ 0, & \text{if} \quad k_{j-1} < k, \text{ or } k > k_{j+1} \end{cases} \quad \lrcorner$$

- Note that $\psi_j(k)$ is the tent function with peak at k_j, and value zero below k_{j-1} and above k_{j+1}.

- **STEP 1**. Divide the state space into a finite number of n disjoint intervals. The points at which the spline pieces are connected are called knots, and there are $n+1$ of these,

$$k_{min} = k_0 < k_1 < k_2 ... < k_i < ... < k_n = k_{max}$$

- **STEP 2**. Guess a solution $\widetilde{V}^s(k)$, that is, guess $\{\widetilde{\theta}_j^s\}_{j=1}^m$, where

$$\widetilde{V}^s(k) = \sum_{j=0}^{m} \widetilde{\theta}_j^s \psi_j(k)$$

and s keeps track of the number of iterations. The first iteration is $s = 0$.

Note that, given $\{\widetilde{\theta}_j^s\}_{j=0}^m$, we can evaluate $\widetilde{V}^s(k_i)$ at any value of capital in the state space, $k \in (k_{min}, k_{max})$, including the capital knots $i = \{0, 1, ..., n\}$.

- **STEP 3**. For each capital knot $i = \{0, 1, ..., n\}$, find

$$g^s(k_i) = \arg \max_{k' \in [k_{min}, k_{max}]} u(f(k_i) + (1 - \delta)k_i - k') + \beta \widetilde{V}^s(k') \tag{7}$$

and store $\{g^s(k_i)\}_{i=0}^n$.

To solve for k' (i.e., find $g^s(k_i)$) in (7) one can:

- implement a numerical optimization algorithm to (7),
- or one can compute the first order condition of (7)—note that we know the value function $\widetilde{V}^s(k')$, and hence we can take numerical derivatives—and solve the for k' in a univariate nonlinear equation.

- **STEP 4**. Given $\{g^s(k_i)\}_{i=0}^n$, update the value function at each $i = 0, 1, ..., n$,

$$\widetilde{V}^{s+1}(k_i) = u(f(k_i) + (1-\delta)k_i - g^s(k_i)) + \beta \widetilde{V}^s(g^s(k_i))$$

and store $\{\widetilde{V}^{s+1}(k_i)\}_{i=0}^n$.

- **STEP 5**. Given $\{\widetilde{V}^{s+1}(k_i)\}_{i=0}^{n}$, solve for the m-basis coefficients $\{\theta_j^{s+1}\}_{j=0}^{m}$ in

$$\widetilde{V}^{s+1}(k_i) = \sum_{j=0}^{m} \theta_j^{s+1} \psi_j(k_i) \quad \forall\, i = \{0, ..., n\} \tag{8}$$

That is, (8) is a nonlinear system with $n+1$ equations and m unknowns.

Two plausible options can arise:

- If $m = n$, solving for $\{\theta_j^{s+1}\}_{j=0}^{m}$ in (8) implies that, in practice, we are implementing a weighted residuals method under collocation, and the solution to the nonlinear system (8) exits and it is hopefully unique.

- We can choose to have more knots than basis functions, that is, $n > m$, hence more equations, $\{k_i\}_{i=0}^{n}$, than unknowns, $\{\theta_j^{s+1}\}_{j=0}^{m}$. In this case we can solve for $\{\theta_j^{s+1}\}_{j=0}^{m}$ in (8) by regression, that is, implementing the weighted residuals method under least squares.

- **STEP 6**: If $||\theta^{s+1} - \theta^s|| < \varepsilon$, stop and report success. Otherwise, go back to the previous step, **STEP 3**, replacing $s = s + 1$.

- **STEP 7**: Being thorough.

    - Check the bounds of the state space are not binding. Otherwise, relax them.
    - Make sure ε is small enough. Reduce ε, redo the VFI, and check your answer does not change. Otherwise, keep reducing ε.
    - Make sure the number of knots for capital, n, is large enough. Increase n, redo the VFI, and check your answer does not change. Otherwise, keep increasing n.
    - Make sure the number of basis functions, m, is large enough. Increase m, redo the VFI, and check your answer does not change. Otherwise, keep increasing m.

VFI with a linear-quadratic approximation V

- So far, we have implemented global methods to approximate the value function at each iteration of the VFI algorithm, either by discretatization or using continuous methods.
- An alternative is to approximate the value function locally at each step of the iteration.
- Interested students should read Javier Díaz-Giménez Chapter in Marimon and Scott (1998).