# Value Function Iteration, Policy Function Iteration and Direct Projection

**Macroeconomics 3:** TA class #2

Andrea Pasqualini

Bocconi University

15 February 2021

# Plan for Today

Objective: **Solve numerically for $V(K)$ and $K'(K)$**

Operating example: Neoclassical Growth Model (deterministic version)

$$V(K) \equiv \max_{C,K'} \frac{C^{1-\gamma}}{1-\gamma} + \beta V(K')$$
$$\text{s.t.} \begin{cases} C + K' \leq K^{\alpha} + (1-\delta)K \\ C, K' > 0 \end{cases}$$

Objects of interest

- Policy functions (for simulations)
- Value function (for welfare analyses)

Three leading methods

1. Value Function Iteration
2. Policy Function Iteration
3. Direct Projection

# Overview of Methods

**Value Function Iteration**

- Relies on the Bellman Equation
- Iterates the contraction mapping **T**
- Guarantees convergence to fixed point $V(K)$

**Policy Function Iteration (a.k.a., Howard Improvement)**

- Relies on a once-and-for-all policy function, allowing direct recovery of $V(K)$ from $u(C)$
- Iterates on the policy functions
- Converges by searching the zero of the Bellman equation at the policy functions

**Direct Projection**

- Relies on first-order conditions
- Searches for zeros of the equations
- Should converge, but may be inaccurate

# A Step Back...

What is $K$? What is $V(K)$? What is $K'(K)$?

A computer has no notion of set density, function continuity, irrational numbers, etc.

| Object | Description | Textbook maths | Computer maths |
|---|---|---|---|
| $\mathcal{K}$ | Domain of value/policy fun. | A subset of $\mathbb{R}$ | A vector of scalars |
| $\mathcal{V}$ | Image of value function | A subset of $\mathbb{R}$ | A vector of scalars |
| $K$ | Point on domain | An element of $\mathcal{K}$ | A scalar |
| $V(K)$ | Point on image of value fun. | $V : \mathcal{K} \to \mathcal{V}$ | A scalar |
| $K'(K)$ | Point on image of policy fun. | $K' : \mathcal{K} \to \mathcal{K}$ | A scalar |

Everything must be a specific number, not abstract objects: need to **calibrate** the model

# . . . and Two Steps Forward (1/2)

**Calibration:** Make the model match some observable quantities in the data

- Not enough time to talk about this here (maybe Macro 4?)
- We will set numbers (almost) randomly, interpret results in relative terms

| Symbol | Meaning | Value |
|---|---|---|
| $\alpha$ | Capital intensity in PF | 0.30 |
| $\beta$ | Discount parameter | 0.95 |
| $\gamma$ | CRRA parameter | 1.50 |
| $\delta$ | Capital depreciation | 0.10 |

$$K^{ss} = \left( \frac{1 - \beta(1-\delta)}{\alpha\beta} \right)^{\frac{1}{\alpha-1}}$$
$$\approx 2.62575$$

# . . . and Two Steps Forward (2/2)

**Set up a grid:** Choose the domain for the problem (i.e., where to "look at")

Maths $\mathcal{K} \subseteq \mathbb{R}$

Computer $\mathcal{K} = [k_1, \ldots, k_n]$


Takeaway: a grid samples points over an abstract set (desirably around an interesting point)

Example

# The VFI algorithm

At iteration $j$, with a guess for $V^{(j)}(K)$

1. For every *potential* choice of $K'$
   - 1.1 Compute the corresponding consumption $C$ using the budget constraint
   - 1.2 Enforce the non-negativity constraints (e.g., with a `NaN`)
   - 1.3 Compute $u(C) + \beta V^{(j)}(K)$

2. Choose the value of $K'$ that maximizes the objective
   - The "max" is $V^{(j+1)}(K)$
   - The "argmax" is $K'^{(j+1)}(K)$

3. Compute $\Delta \equiv \max\left(\left|V^{(j+1)}(K) - V^{(j)}(K)\right|\right)$
   - If $\Delta \geq \varepsilon$, repeat 1–4 using $V^{(j+1)}(K)$ as a new guess
   - If $\Delta < \varepsilon$, finish

▸ Into the Numerical Objects

# Remarks

**Pros**

- Surely converges, always
- Relatively easy to code
- Returns the value function (useful for welfare analyses)
- Works for finite-horizon problems, discrete choice models, etc.

**Cons**

- Suffers from the *Curse of Dimensionality*
- Possibly the slowest method of all

**Observations**

- The maximization step takes time
- Can parallelize: algorithm for finding $K'(k_i)$ is independent of $K'(k_j)$

# Policy Function Iteration (a.k.a., Howard Improvement)

**Observation:** at the "true" policy functions $C(K)$ and $K'(K)$, $V(\cdot)$ satisfies

$$V(K) = u(C(K)) + \beta V(K'(K))$$

Looking in terms of vectors and matrices

$$\mathbf{V} = u(\mathbf{C}) + \beta \mathbf{Q} \mathbf{V}$$

$$\mathbf{V} = [\mathbf{I} - \beta \mathbf{Q}]^{-1} u(\mathbf{C}) = \sum_{t=0}^{\infty} \beta^t u(C_t^*)$$

What is $\mathbf{Q}$? A sparse matrix regulating transitions from $K$ to $K' \ldots$ related to $K'(K)$

$$Q_{i,j} = 1\left(K = k_i \wedge K' = k_j\right)$$

# The PFI algorithm

At iteration $j$, with a guess for $K'^{(j)}(K)$

1. Back out the the implied value function $\mathbf{V}$
   - 1.1 Derive the matrix $\mathbf{Q}$ from the guess $K'^{(j)}(K)$
   - 1.2 Compute consumption $C$ from the budget constraint
   - 1.3 Enforce non-negativity constraints
   - 1.4 Compute $\mathbf{V} = [\mathbf{I} - \beta\mathbf{Q}]^{-1} u(\mathbf{C})$
2. For every *potential* choice of $K'$
   - 2.1 Compute consumption $C$ from the budget constraints
   - 2.2 Enforce non-negativity constraints
   - 2.3 Compute $u(C) + \beta\mathbf{V}$
3. Choose the value $K'$ that maximizes the objective
   - The "argmax" is $K'^{(j+1)}(K)$
4. Compute $\Delta \equiv \max\left(\left|K'^{(j+1)}(K) - K'^{(j)}(K)\right|\right)$
   - If $\Delta \geq \varepsilon$, repeat 2–5 using $K'^{(j+1)}(K)$ as a new guess
   - If $\Delta < \varepsilon$, finish

# Remarks about PFI

**Pros**

- Convergence has been proven, always
- Returns the value function (useful for welfare analyses)
- Is faster than VFI if the contraction modulus of $\mathbf{T}$ is close to unity

**Cons**

- Suffers from the *Curse of Dimensionality*
- May be slower than VFI, difficult to check ex-ante
- The initial condition may matter
- Infeasible for time-varying policy functions, discrete choice models, finite-horizon problems

**Observations**

- The maximization step takes time
- Inverting $[\mathbf{I} - \beta\mathbf{Q}]$ may be computationally expensive
- Can parallelize the maximization step for each $K$

# Direct Projection

Compute the first-order conditions of the maximization problem

$$\begin{cases} C(K) = C(K'(K))\left(\beta\alpha K'(K)^{\alpha-1} + 1 - \delta\right)^{-\frac{1}{\gamma}} \\ C(K) + K'(K) = K^{\alpha} + (1-\delta)K \end{cases}$$

- Solve for $C(K)$: Can ask the computer the find the zero of the (functional) Euler equation
- Solve for $K'(K)$: Use the second equation

**Catch:** These are *functional* equations

- Choosing $C(K)\ldots$ meaning, choosing **C**
- $\ldots$ implies choosing $K'(K)$ from the budget constraint$\ldots$ meaning, obtaining $\mathbf{K'}$
- $\ldots$ which implies something for $C(K'(K))$ $\ldots$ no way of doing this numerically!

**Workaround:** Given an initial condition, interpolate $C(K)$ and recompute points on $C(K'(K))$

# Remarks about Direct Projection

**Pros**

- Zero-finding routine easily available in `scipy`
- May be faster than VFI/PFI

**Cons**

- Convergence **not** guaranteed
- May be numerically inaccurate

**Observations**

- Cannot be parallelized (if it uses interpolation over the grid)
- Checking accuracy requires solving with alternative methods...

# Practice time

Moving to a Jupyter Notebook

# Exercises

1. Use the code I showed you for VFI, keeping all the calibration values and parameters
   - 1.1 Plot the no. of iterations required for convergence against the no. of points on the grid (e.g., $N \in \{50, 100, 150, \ldots, 1000\}$)
   - 1.2 Plot the no. of iterations required for convergence against the value of $\beta$ (e.g., $\beta \in [0.75, 1)$)
   - 1.3 What can you say about speed of convergence of the VFI algorithm?

2. Write your own code to implement Howard's Improvement (PFI) as in the slide above (i.e., do not use the shortcut I showed in the practice code in class)
   - 2.1 Plot the no. of iterations required for convergence against the value of $\beta$ (e.g., $\beta \in [0.75, 1)$)
   - 2.2 What can you say about the speed of PFI relative to the speed of VFI?

3. Use the code I showed you for the Direct Projection method. Change the initial condition for the policy function with arbitrary choices. What can you say about the sensitivity of the algorithm w.r.t. the initial condition?

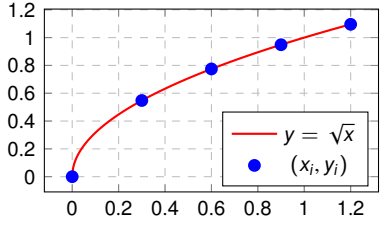
4. In what sense does any of the methods of this class deliver an approximation of the "true" value function?

# Vectors, Domains and Functions in Computers

Consider $f : X \to Y$, with $X, Y \subseteq \mathbb{R}$, such that $y = \sqrt{x}$ (i.e., $f(\cdot) \equiv \sqrt{\cdot}$)

$$X \equiv \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}' \qquad Y \equiv \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}' \qquad \text{with } y_i = \sqrt{x_i},\ \forall\ i \in \{1, 2, \ldots, n\}$$

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(0, 1.2, num=5)
Y = np.sqrt(X)
plt.plot(X, Y)
```




Back

# Peeking into Numerical Objects with VFI

Convention: rows are states, columns are controls

The budget constraint

$$\underbrace{\begin{bmatrix} k_1^\alpha \\ k_2^\alpha \\ \vdots \\ k_n^\alpha \end{bmatrix}}_{K^\alpha} + (1-\delta) \underbrace{\begin{bmatrix} k_1^\alpha \\ k_2^\alpha \\ \vdots \\ k_n^\alpha \end{bmatrix}}_{K} - \underbrace{\begin{bmatrix} k_1 & k_2 & \cdots & k_n \end{bmatrix}}_{K'} = \underbrace{\begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix}}_{C}$$

The value function (to be maximized—keep best element in each row)

$$\underbrace{\begin{bmatrix} \tilde{V}_{1,1} & \tilde{V}_{1,2} & \cdots & \tilde{V}_{1,n} \\ \tilde{V}_{2,1} & \tilde{V}_{2,2} & \cdots & \tilde{V}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{V}_{n,1} & \tilde{V}_{n,2} & \cdots & \tilde{V}_{n,n} \end{bmatrix}}_{\tilde{V}(K,K')} = \underbrace{\begin{bmatrix} u(c_{1,1}) & u(c_{1,2}) & \cdots & u(c_{1,n}) \\ u(c_{2,1}) & u(c_{2,2}) & \cdots & u(c_{2,n}) \\ \vdots & \vdots & \ddots & \vdots \\ u(c_{n,1}) & u(c_{n,2}) & \cdots & u(c_{n,n}) \end{bmatrix}}_{u(C)} + \beta \underbrace{\begin{bmatrix} V(k_1) & V(k_2) & \cdots & V(k_n) \end{bmatrix}}_{V(K')}$$

(Linear algebra doubts? Broadcasting!)