Quantitative Macroeconomics I
TD 3: Value Function Iteration

Grégoire Sempé

gregoire.sempe@psemail.eu

Paris School of Economics, Université Paris 1 Panthéon-Sorbonne

September 29, 2025

Today's Agenda

1. Feedback & questions about Problem Set I

2. **Theory**: Bellman Equation, example of a consumption-saving program (reminder)
   - Recursive form of the deterministic problem
   - Markov chains and stochastic dynamic programming
   - Contraction mapping theorem and backward iteration

3. **Computational**: Value Function Iteration

   Global method, on the state space, can study uncertainty
   - On-grid Value Function Iteration
   - Off-grid VFI & Euler errors

   $\Rightarrow$ Pseudo-code of the algorithm (whiteboard)

A consumption saving program, without uncertainty

For the PS, you will be asked to solve a Real Business Cycles model. In this tutorial, we will take the example of a consumption (c_t) saving (a_{t+1}) program in partial eq. You will have to think carefully about the differences between the two models for the problem set…

A consumption saving program, without uncertainty

For the PS, you will be asked to solve a Real Business Cycles model. In this tutorial, we will take the example of a consumption (c_t) saving (a_{t+1}) program in partial eq. You will have to think carefully about the differences between the two models for the problem set…

Challenge: Find the sequence of $\{c_t, a_{t+1}\}_{\forall s \geq t}$ that solves:

A consumption saving program, without uncertainty

For the PS, you will be asked to solve a Real Business Cycles model. In this tutorial, we will take the example of a consumption (c_t) saving (a_{t+1}) program in partial eq. You will have to think carefully about the differences between the two models for the problem set…

Challenge: Find the sequence of $\{c_t, a_{t+1}\}_{\forall s \geq t}$ that solves:

$$V_t(a_t) \equiv \max_{\{c_s, a_{s+1}\}_{\forall s \geq t}} \sum_{s \geq t} \beta^{s-t} u(c_s) \quad \text{s.t.} \quad \begin{cases} c_s + a_{s+1} = (1 + r_s) a_s + \bar{y} & \forall s \geq t \\ a_s \geq 0 & \forall s \geq t \\ a_t \quad \text{is given} & \end{cases}$$

A consumption saving program, without uncertainty

For the PS, you will be asked to solve a Real Business Cycles model. In this tutorial, we will take the example of a consumption (c_t) saving (a_{t+1}) program in partial eq. You will have to think carefully about the differences between the two models for the problem set...

Challenge: Find the sequence of $\{c_t, a_{t+1}\}_{\forall s \geq t}$ that solves:

$$V_t(a_t) \equiv \max_{\{c_s, a_{s+1}\}_{\forall s \geq t}} \sum_{s \geq t} \beta^{s-t} u(c_s) \quad \text{s.t.} \quad \begin{cases} c_s + a_{s+1} = (1 + r_s)a_s + \bar{y} & \forall s \geq t \\ a_s \geq 0 & \forall s \geq t \\ a_t \quad \text{is given} & \end{cases}$$

Issue: This problem is subject to the curse of dimensionality...

$\rightarrow$ How would you reduce its dimensionality to find a global solution?

Dimensionality reduction of the problem

1/ Algebra: reduce the number of control variables using the budget constraint

$$V_t(a_t) = \max_{\{a_{s+1}\}_{\forall s \geq t}} \sum_{s \geq t} \beta^{s-t} u\left((1+r_s)a_s + \bar{y} - a_{s+1}\right) \quad \text{s.t.} \quad a_{s+1} \geq 0 \quad \forall s \geq t$$

Dimensionality reduction of the problem

1/ Algebra: reduce the number of control variables using the budget constraint

$$V_t(a_t) = \max_{\{a_{s+1}\}_{\forall s \geq t}} \sum_{s \geq t} \beta^{s-t} u\left((1+r_s)a_s + \bar{y} - a_{s+1}\right) \quad \text{s.t.} \quad a_{s+1} \geq 0 \quad \forall s \geq t$$

2/ We can write the problem in the **state space** (recursive form)

$$V_t(a_t) = \max_{a_{t+1}} \left\{ u(c_t) + \beta \underbrace{\left[\max_{\{a_{s+1}\}_{\forall s \geq t+1}} u(c_{t+1}) + \sum_{s \geq t+1} \beta^{s-(t+1)} u(c_s) \right]}_{V_{t+1}(a_{t+1})} \right\}$$

$$\text{s.t.} \quad c_t = (1+r_t)a_t + \bar{y} - a_{t+1} \qquad \text{and} \quad a_{t+1} \geq 0$$

The deterministic Bellman Equation

Bellman Equation reduces the problem to ”today’s choice” given ”tomorrow” optimal

$$V_t(a_t) = \underbrace{\mathcal{T}\{V_{t+1}\}(a_t)}_{\text{Bellman operator}} = \max_{a_{t+1}} u\left((1+r)a_t + \bar{y} - a_{t+1}\right) + \beta V_{t+1}(a_{t+1}) \qquad \text{s.t.} \quad a_{t+1} \geq 0$$

The deterministic Bellman Equation

Bellman Equation reduces the problem to ”today’s choice” given ”tomorrow” optimal

$$V_t(a_t) = \underbrace{\mathcal{T}\{V_{t+1}\}(a_t)}_{\text{Bellman operator}} = \max_{a_{t+1}} u\left((1+r)a_t + \bar{y} - a_{t+1}\right) + \beta V_{t+1}(a_{t+1}) \qquad \text{s.t.} \quad a_{t+1} \geq 0$$

1. Problem is to find the optimal **policy function** $a_{t+1} = g_t(a_t)$, instead of a sequence

   $\Rightarrow$ Solution to the curse of dimensionality!

The deterministic Bellman Equation

Bellman Equation reduces the problem to ”today’s choice” given ”tomorrow” optimal

$$V_t(a_t) = \underbrace{\mathcal{T}\{V_{t+1}\}(a_t)}_{\text{Bellman operator}} = \max_{a_{t+1}} u\left((1+r)a_t + \bar{y} - a_{t+1}\right) + \beta V_{t+1}(a_{t+1}) \qquad \text{s.t.} \quad a_{t+1} \geq 0$$

1. Problem is to find the optimal **policy <u>function</u>** $a_{t+1} = g_t(a_t)$, instead of a sequence

   $\Rightarrow$ Solution to the curse of dimensionality!

2. $V_t(a_t)$ is the discounted sum of utility, given states and with optimal policies (Value Function)

The deterministic Bellman Equation

Bellman Equation reduces the problem to "today's choice" given "tomorrow" optimal

$$V_t(a_t) = \underbrace{\mathcal{T}\{V_{t+1}\}(a_t)}_{\text{Bellman operator}} = \max_{a_{t+1}} u\left((1+r)a_t + \bar{y} - a_{t+1}\right) + \beta V_{t+1}(a_{t+1}) \qquad \text{s.t.} \quad a_{t+1} \geq 0$$

1. Problem is to find the optimal **policy <u>function</u>** $a_{t+1} = g_t(a_t)$, instead of a sequence

   $\Rightarrow$ Solution to the curse of dimensionality!

2. $V_t(a_t)$ is the discounted sum of utility, given states and with optimal policies (Value Function)

3. $a_{t+1} \in \Gamma(a_t)$ is the **Choice correspondence**

The deterministic Bellman Equation

Bellman Equation reduces the problem to "today's choice" given "tomorrow" optimal

$$V_t(a_t) = \underbrace{\mathcal{T}\{V_{t+1}\}(a_t)}_{\text{Bellman operator}} = \max_{a_{t+1}} u\left((1+r)a_t + \bar{y} - a_{t+1}\right) + \beta V_{t+1}(a_{t+1}) \qquad \text{s.t.} \quad a_{t+1} \geq 0$$

1. Problem is to find the optimal **policy <u>function</u>** $a_{t+1} = g_t(a_t)$, instead of a sequence
   $\Rightarrow$ Solution to the curse of dimensionality!
2. $V_t(a_t)$ is the discounted sum of utility, given states and with optimal policies (Value Function)
3. $a_{t+1} \in \Gamma(a_t)$ is the **Choice correspondence**
4. The Bellman Equation maps a function into a function. It is a ***functional equation***

Deriving the Euler Equation using the Bellman Equation

$$V_t(a_t) = \max_{c_t, a_{t+1}} \left\{ u(c_t) + \beta V_{t+1}(a_{t+1}) \right\} \quad \& \quad a_{t+1} = \bar{y} + R \times a_t - c_t$$

Deriving the Euler Equation using the Bellman Equation

$$V_t(a_t) = \max_{c_t, a_{t+1}} \left\{ u(c_t) + \beta V_{t+1}(a_{t+1}) \right\} \quad \& \quad a_{t+1} = \bar{y} + R \times a_t - c_t$$

1. Taking the FOC on c_t yields

$$u'(c_t) + \beta \times \frac{\partial V_{t+1}}{\partial a_{t+1}} \underbrace{\frac{\partial a_{t+1}}{\partial c_t}}_{=-1} = 0 \iff u'(c_t) = \beta \times \frac{\partial V_{t+1}}{\partial a_{t+1}}$$

Deriving the Euler Equation using the Bellman Equation

$$V_t(a_t) = \max_{c_t, a_{t+1}} \left\{ u(c_t) + \beta V_{t+1}(a_{t+1}) \right\} \qquad \& \qquad a_{t+1} = \bar{y} + R \times a_t - c_t$$

1. Taking the FOC on c_t yields

$$u'(c_t) + \beta \times \frac{\partial V_{t+1}}{\partial a_{t+1}} \underbrace{\frac{\partial a_{t+1}}{\partial c_t}}_{=-1} = 0 \iff u'(c_t) = \beta \times \frac{\partial V_{t+1}}{\partial a_{t+1}}$$

2. Use the Envelope theorem

   More details: `https://www.econ2.jhu.edu/people/ccarroll/public/lecturenotes/consumption/Envelope/`

$$\frac{\partial V_t}{\partial a_t} = \frac{\partial}{\partial a_t} \Big[u((1+r)a_t + \bar{y} - a_{t+1}) + \beta V_{t+1}(a_{t+1}) \Big] \Big|_{a_{t+1} = a^*_{t+1}(a_t)} = \frac{\partial u(c_t)}{\partial c_t} \times \frac{\partial c_t}{\partial a_t} = R \times u'(c_t)$$

Deriving the Euler Equation using the Bellman Equation

$$V_t(a_t) = \max_{c_t, a_{t+1}} \left\{ u(c_t) + \beta V_{t+1}(a_{t+1}) \right\} \quad \& \quad a_{t+1} = \bar{y} + R \times a_t - c_t$$

1. Taking the FOC on c_t yields

$$u'(c_t) + \beta \times \frac{\partial V_{t+1}}{\partial a_{t+1}} \underbrace{\frac{\partial a_{t+1}}{\partial c_t}}_{=-1} = 0 \iff u'(c_t) = \beta \times \frac{\partial V_{t+1}}{\partial a_{t+1}}$$

2. Use the Envelope theorem

   More details: `https://www.econ2.jhu.edu/people/ccarroll/public/lecturenotes/consumption/Envelope/`

$$\frac{\partial V_t}{\partial a_t} = \frac{\partial}{\partial a_t} \left[u((1+r)a_t + \bar{y} - a_{t+1}) + \beta V_{t+1}(a_{t+1}) \right] \Bigg|_{a_{t+1} = a^*_{t+1}(a_t)} = \frac{\partial u(c_t)}{\partial c_t} \times \frac{\partial c_t}{\partial a_t} = R \times u'(c_t)$$

3. Combining both yields the **Euler Equation**:

$$u'(c_t) = \beta(1+r)u'(c_{t+1})$$

Adding risk: Markov chains

Risk is a key feature of economic behavior, often modeled using **Markov processes**

$\hookrightarrow$ In our example, take households' earning, subject to risk $\omega \sim AR(1)$

Formally, uncertainty is different from risk. It is often modelled as a risk on the variance of the AR1 process. Ask Moritz to know more!

Adding risk: Markov chains

Risk is a key feature of economic behavior, often modeled using **Markov processes**

$\hookrightarrow$ In our example, take households' earning, subject to risk $\omega \sim AR(1)$

1/ Markov processes have desirable properties for dynamic programming

Formally, uncertainty is different from risk. It is often modelled as a risk on the variance of the AR1 process. Ask Moritz to know more!

Adding risk: Markov chains

Risk is a key feature of economic behavior, often modeled using **Markov processes**
$\hookrightarrow$ In our example, take households' earning, subject to risk $\omega \sim AR(1)$

1/ Markov processes have desirable properties for dynamic programming

   (a) **Memoryless**: "Future states depend only on the current state"

Formally, uncertainty is different from risk. It is often modelled as a risk on the variance of the AR1 process. Ask Moritz to know more!

Adding risk: Markov chains

Risk is a key feature of economic behavior, often modeled using **Markov processes**
$\hookrightarrow$ In our example, take households' earning, subject to risk $\omega \sim AR(1)$

1/ Markov processes have desirable properties for dynamic programming

   (a) **Memoryless**: "Future states depend only on the current state"
   (b) **Stationary transitions**: Transition probabilities are time-invariant

Formally, uncertainty is different from risk. It is often modelled as a risk on the variance of the AR1 process. Ask Moritz to know more!

Adding risk: Markov chains

Risk is a key feature of economic behavior, often modeled using **Markov processes**
$\hookrightarrow$ In our example, take households' earning, subject to risk $\omega \sim AR(1)$

1/ Markov processes have desirable properties for dynamic programming

(a) **Memoryless**: "Future states depend only on the current state"

(b) **Stationary transitions**: Transition probabilities are time-invariant

(c) **Discretization**: Approximate continuous processes by markov chains (Tauchen, Rowenhorst)

Formally, uncertainty is different from risk. It is often modelled as a risk on the variance of the AR1 process. Ask Moritz to know more!

Adding risk: Markov chains

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

Risk is a key feature of economic behavior, often modeled using **Markov processes**

$\hookrightarrow$ In our example, take households' earning, subject to risk $\omega \sim AR(1)$

1/ Markov processes have desirable properties for dynamic programming

(a) **Memoryless**: "Future states depend only on the current state"

(b) **Stationary transitions**: Transition probabilities are time-invariant

(c) **Discretization**: Approximate continuous processes by markov chains (Tauchen, Rowenhorst)

2/ Markov chains are defined by

- A discrete set of states Ω, with a probability transition matrix $\Pi = (\pi_{\omega,\omega'})_{\forall \omega,\omega' \in \Omega^2}$
- An initial distribution μ_0

Formally, uncertainty is different from risk. It is often modelled as a risk on the variance of the AR1 process. Ask Moritz to know more!

Stochastic Bellman Equation

Earnings are stochastic $\rightarrow$ replace $\bar{y}$ by ω. The Bellman Equation becomes

$$
\begin{aligned}
V_t(\omega_t, a_t) &= \mathcal{T}\{V_{t+1}\}(\omega_t, a_t) \\
&= \max_{a_{t+1}} u\left((1+r)a_t + \omega_t - a_{t+1}\right) + \beta \mathbb{E}_{\omega_{t+1}|\omega_t} V_{t+1}(\omega_{t+1}, a_{t+1}) \\
&= \max_{a_{t+1}} u\left((1+r)a_t + \omega_t - a_{t+1}\right) + \beta \sum_{\omega_{t+1}} \pi_{\omega_t,\omega_{t+1}} V_{t+1}(\omega_{t+1}, a_{t+1}) \\
\text{s.t.} \quad & a_{t+1} \geq 0 \qquad \forall\ \omega_t, a_t \in \Omega \times \mathbb{R}_+
\end{aligned}
$$

Two useful applications of the Bellman Equation

1. **At steady state**, the Value Function is the unique[1] fixed point to the Bellman operator

$$\text{CMT} : V^* \text{ s.t. } V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a) = \max_{a'} u\left((1+r)a + \omega - a'\right) + \beta \mathbb{E}_{\omega'|\omega} V^*(\omega', a')$$

[1] By the application of the contraction mapping theorem. See the lecture slides.

Two useful applications of the Bellman Equation

1. **At steady state**, the Value Function is the unique[1] fixed point to the Bellman operator

$$\text{CMT}: V^* \text{ s.t. } V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a) = \max_{a'} u\left((1+r)a + \omega - a'\right) + \beta \mathbb{E}_{\omega'|\omega} V^*(\omega', a')$$

2. **Backward induction** (in finite time):

   1/ Start from a terminal condition $V_{T+1}(a_{T+1})$ e.g $V_{T+1}(a_{T+1}) = 0$ if T is large enough, HH is dead after T

   2/ Given a sequence of ω_t, the Value function at time $t < T+1$ is obtained by applying the Bellman Operator

$$V_t(\omega_t, a_t) = \mathcal{T}\{V_{t+1}\}(\omega_t, a_t) \qquad \forall a_t$$

[1] By the application of the contraction mapping theorem. See the lecture slides.

Two useful applications of the Bellman Equation

1. **At steady state**, the Value Function is the unique[1] fixed point to the Bellman operator

$$\text{CMT}: V^* \text{ s.t. } V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a) = \max_{a'} u\left((1+r)a + \omega - a'\right) + \beta \mathbb{E}_{\omega'|\omega} V^*(\omega', a')$$

2. **Backward induction** (in finite time):

   1/ Start from a terminal condition $V_{T+1}(a_{T+1})$ e.g $V_{T+1}(a_{T+1}) = 0$ if T is large enough, HH is dead after T

   2/ Given a sequence of ω_t, the Value function at time $t < T+1$ is obtained by applying the Bellman Operator

$$V_t(\omega_t, a_t) = \mathcal{T}\{V_{t+1}\}(\omega_t, a_t) \qquad \forall a_t$$

$\Rightarrow$ We can find the sequence of optimal policy functions $\{a'_t(a_t)\}_{t=0}^{T}$

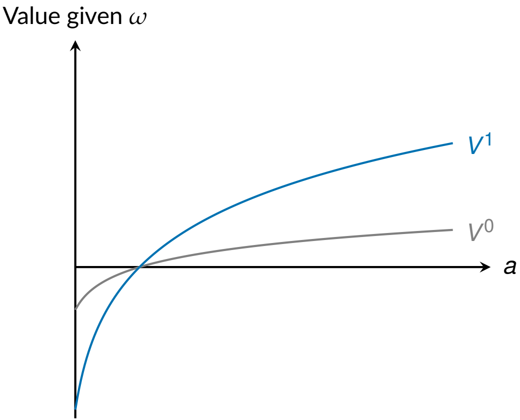
[1] By the application of the contraction mapping theorem. See the lecture slides.

Stationary Value Function Iteration (VFI)

How to find the value function and policy functions at steady state?

Stationary Value Function Iteration (VFI)

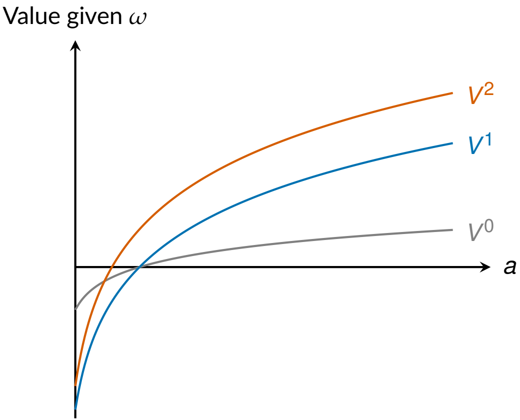
How to find the value function and policy functions at steady state?



- Guess an initial value function $V^0(\omega, a)$
- New guess: $V^1(\omega, a) = \mathcal{T}\{V^0\}(\omega, a)$

Stationary Value Function Iteration (VFI)

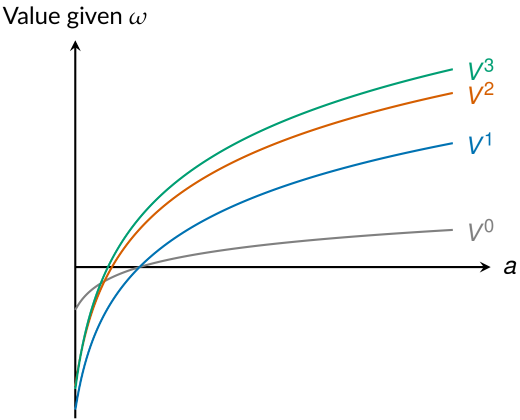
How to find the value function and policy functions at steady state?



- Guess an initial value function $V^0(\omega, a)$
- New guess: $V^1(\omega, a) = \mathcal{T}\{V^0\}(\omega, a)$
- Continue: $V^2(\omega, a) = \mathcal{T}\{V^1\}(\omega, a)$

Stationary Value Function Iteration (VFI)

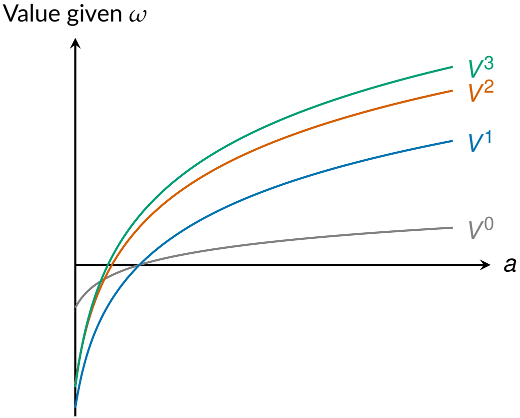
How to find the value function and policy functions at steady state?



- Guess an initial value function $V^0(\omega, a)$
- New guess: $V^1(\omega, a) = \mathcal{T}\{V^0\}(\omega, a)$
- Continue: $V^2(\omega, a) = \mathcal{T}\{V^1\}(\omega, a)$
- Again: $V^3(\omega, a) = \mathcal{T}\{V^2\}(\omega, a)$

Stationary Value Function Iteration (VFI)

How to find the value function and policy functions at steady state?



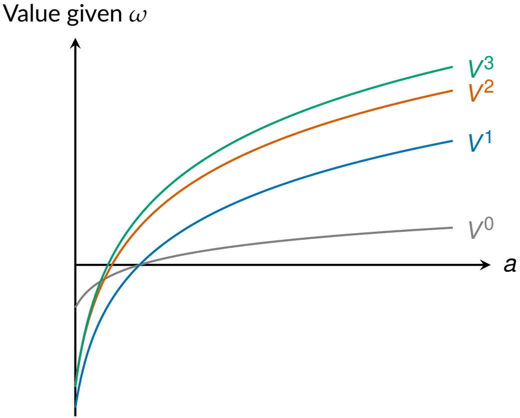
- Guess an initial value function $V^0(\omega, a)$
- New guess: $V^1(\omega, a) = \mathcal{T}\{V^0\}(\omega, a)$
- Continue: $V^2(\omega, a) = \mathcal{T}\{V^1\}(\omega, a)$
- Again: $V^3(\omega, a) = \mathcal{T}\{V^2\}(\omega, a)$

... Continue until "convergence" btw. functions

$$\|V^n - V^{(n-1)}\| \leq \varepsilon$$

Stationary Value Function Iteration (VFI)

How to find the value function and policy functions at steady state?



- Guess an initial value function $V^0(\omega, a)$
- New guess: $V^1(\omega, a) = \mathcal{T}\{V^0\}(\omega, a)$
- Continue: $V^2(\omega, a) = \mathcal{T}\{V^1\}(\omega, a)$
- Again: $V^3(\omega, a) = \mathcal{T}\{V^2\}(\omega, a)$

... Continue until "convergence" btw. functions

$$\|V^n - V^{(n-1)}\| \leq \varepsilon$$

$\Longrightarrow$ Let's do a pseudo-code of the algorithm together!

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a) \rightarrow$

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a)$ $\rightarrow$ How can we store a function?

Pseudo-code for stationary VFI – Roadmap

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a) \rightarrow$ How can we store a function?
3. We need to code the **Bellman operator** $\mathcal{T} \rightarrow$

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a) \rightarrow$ How can we store a function?
3. We need to code the **Bellman operator** $\mathcal{T} \rightarrow$ How?

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a)$ $\rightarrow$ How can we store a function?
3. We need to code the **Bellman operator** $\mathcal{T}$ $\rightarrow$ How?
4. We need to iterate until a convergence criterion/measure is met $\rightarrow$

Pseudo-code for stationary VFI – Roadmap

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a)$ $\rightarrow$ How can we store a function?
3. We need to code the **Bellman operator** $\mathcal{T}$ $\rightarrow$ How?
4. We need to iterate until a convergence criterion/measure is met $\rightarrow$ kind of norm?

Pseudo-code for stationary VFI – Roadmap

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

Stationary Value Function Iteration

Goal: Find the fixed point of the Bellman equation $V^*(\omega, a) = \mathcal{T}\{V^*\}(\omega, a)$

1. We need to define the **parameters** of the model $\rightarrow$ easy! use a structure array
2. We need to initialize the value function $V^0(\omega, a)$ $\rightarrow$ How can we store a function?
3. We need to code the **Bellman operator** $\mathcal{T}$ $\rightarrow$ How?
4. We need to iterate until a convergence criterion/measure is met $\rightarrow$ kind of norm?
5. We need to store results (VF, PF), and check if they make sense $\rightarrow$ Plotting, Euler errors?

Storing Value Functions and Policy Functions in MATLAB

VFI consists in iterating on the value function by applying the Bellman operator

- At each iteration, we need to **store the value function**

  Remember, V : state space $\rightarrow \mathbb{R}$, but the state space is continuous!

  $\Rightarrow$ Solution: **discretize** the state space into **grids** of points $(\omega_i, a_j) \in \mathcal{G}_\omega \times \mathcal{G}_a$

Note: You could also use projection methods to store these functions (e.g. Chebyshev polynomials), see Tobias' slides

Storing Value Functions and Policy Functions in MATLAB

VFI consists in iterating on the value function by applying the Bellman operator

- At each iteration, we need to **store the value function**

  Remember, V : state space $\rightarrow \mathbb{R}$, but the state space is continuous!

  $\Rightarrow$ Solution: **discretize** the state space into **grids** of points $(\omega_i, a_j) \in \mathcal{G}_\omega \times \mathcal{G}_a$

- The value function is stored as a **matrix** $\mathbf{V} = (V(\omega_i, a_j))_{\forall i,j}$ (size: $N_\omega \times N_a$)

  The same holds for the policy functions!

Note: You could also use projection methods to store these functions (e.g. Chebyshev polynomials), see Tobias' slides

Storing Value Functions and Policy Functions in MATLAB

VFI consists in iterating on the value function by applying the Bellman operator

- At each iteration, we need to **store the value function**

  Remember, V : state space $\to \mathbb{R}$, but the state space is continuous!

  $\Rightarrow$ Solution: **discretize** the state space into **grids** of points $(\omega_i, a_j) \in \mathcal{G}_\omega \times \mathcal{G}_a$

- The value function is stored as a **matrix V** $= (V(\omega_i, a_j))_{\forall i,j}$ (size: $N_\omega \times N_a$)

  The same holds for the policy functions!

$\Rightarrow$ Convergence between two matrices (*discretized value functions*) using a norm

$$||\cdot||_\infty = \max_{i,j} |V^{(2)}(\omega_i, a_j) - V^{(1)}(\omega_i, a_j)| \qquad \text{with} \quad \omega_i, a_j \in \mathcal{G}_\omega \times \mathcal{G}_a$$

Note: You could also use projection methods to store these functions (e.g. Chebyshev polynomials), see Tobias' slides

Coding the Bellman operator $\mathcal{T}$

1/ The state space is discretized, but the choice space is continuous

- ω and a are states, discretized on grids $\mathcal{G}_\omega$ and $\mathcal{G}_a$
- a' is a choice, **continuous** on $\mathbb{R}_+$

Coding the Bellman operator $\mathcal{T}$

1/ The state space is discretized, but the choice space is continuous

- ω and a are states, discretized on grids $\mathcal{G}_\omega$ and $\mathcal{G}_a$
- a' is a choice, **continuous** on $\mathbb{R}_+$

2/ At each iteration, we need to solve the max problem **for** each point of the gridded state space

$$\forall \omega_i, a_j \in \mathcal{G}_\omega \times \mathcal{G}_a \qquad V(\omega_i, a_j) = \max_{a' \in \mathbb{R}_+} u\left((1+r)a_j + \omega_i - a'\right) + \beta \sum_{\omega'} \pi_{\omega_i,\omega'} V^{\text{prev}}(\omega', a')$$

Coding the Bellman operator $\mathcal{T}$

PARIS SCHOOL OF ECONOMICS
ÉCOLE D'ÉCONOMIE DE PARIS

1/ The state space is discretized, but the choice space is continuous

- ω and a are states, discretized on grids $\mathcal{G}_\omega$ and $\mathcal{G}_a$
- a' is a choice, **continuous** on $\mathbb{R}_+$

2/ At each iteration, we need to solve the max problem **for** each point of the gridded state space

$$\forall \omega_i, a_j \in \mathcal{G}_\omega \times \mathcal{G}_a \qquad V(\omega_i, a_j) = \max_{a' \in \mathbb{R}_+} u\left((1+r)a_j + \omega_i - a'\right) + \beta \sum_{\omega'} \pi_{\omega_i,\omega'} V^{\text{prev}}(\omega', a')$$

3/ How to solve this problem numerically? Two options:

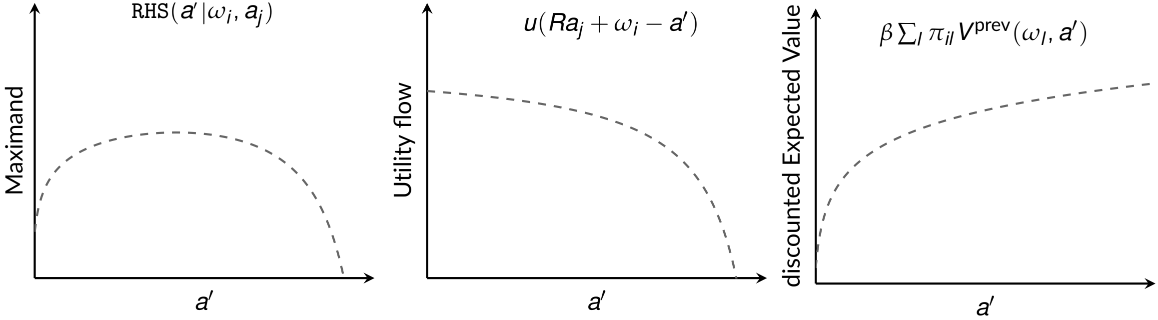
(a) On-grid VFI: restrict a' to be on the grid $\mathcal{G}_a$ (grid search) $\rightarrow$ **extremely imprecise**

(b) Off-grid VFI: use a numerical optimizer (e.g. golden-section search), **interpolate** between grid points

Diving into the RHS of the Bellman – given $\omega = \omega_i$ & $a = a_j$

Maximand of the RHS of the Bellman Equation

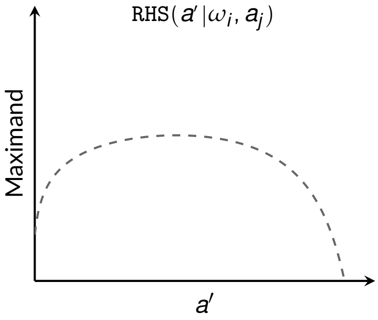
$$\text{RHS}(a'|\omega_i, a_j) = u(Ra_j + \omega_i - a') + \beta \sum_{\omega'} \pi_{\omega_i,\omega'} V^{\text{prev}}(\omega', a')$$

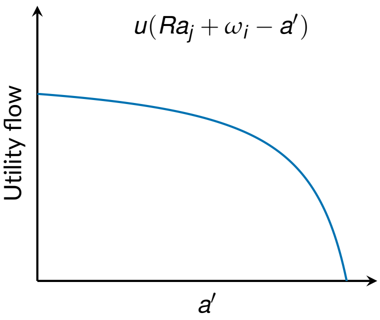


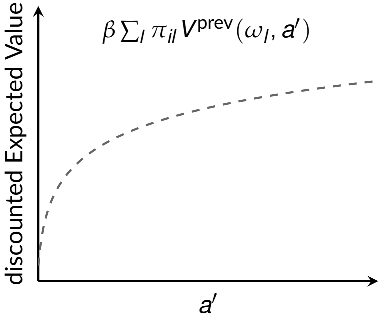
Diving into the RHS of the Bellman – given $\omega = \omega_i$ & $a = a_j$

Maximand of the RHS of the Bellman Equation

$$\text{RHS}(a'|\omega_i, a_j) = \underbrace{u(Ra_j + \omega_i - a')}_{\text{Closed form}} + \underbrace{\beta \sum_{\omega'} \pi_{\omega_i,\omega'}}_{\text{weighted sum}} V^{\text{prev}}(\omega', a')$$



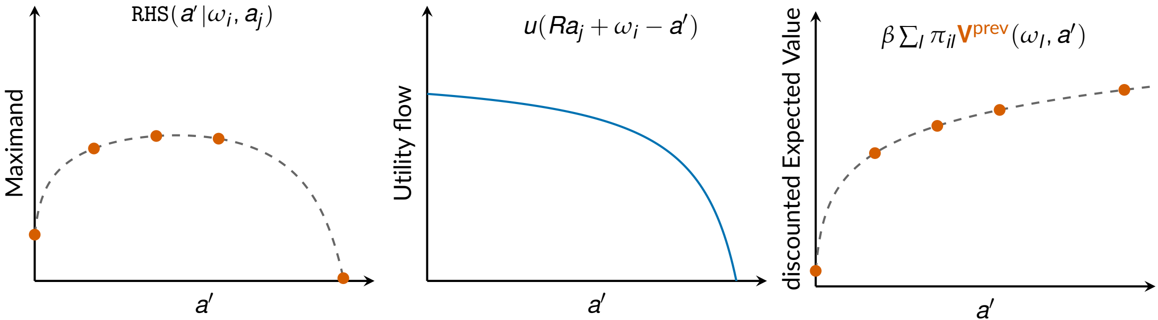




Diving into the RHS of the Bellman – given $\omega = \omega_i$ & $a = a_j$

Maximand of the RHS of the Bellman Equation

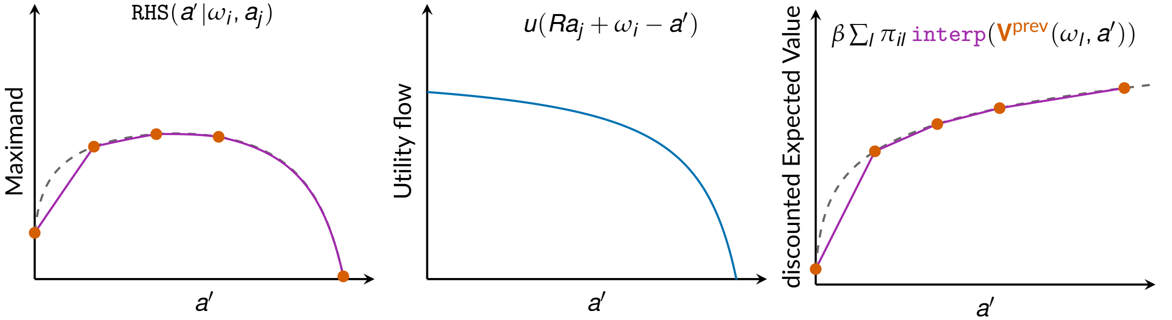
$$\mathrm{RHS}(a'|\omega_i, a_j) = \underbrace{u(Ra_j + \omega_i - a')}_{\text{Closed form}} + \underbrace{\beta \sum_{\omega'} \pi_{\omega_i,\omega'}}_{\text{weighted sum}} \mathbf{V}^{\text{prev}}(\omega', a')$$



Diving into the RHS of the Bellman – given $\omega = \omega_i$ & $a = a_j$

Maximand of the RHS of the Bellman Equation

$$\text{RHS}(a'|\omega_i, a_j) = \underbrace{u(Ra_j + \omega_i - a')}_{\text{Closed form}} + \underbrace{\beta \sum_{\omega'} \pi_{\omega_i,\omega'}}_{\text{weighted sum}} \underbrace{\texttt{interp}\left(\mathbf{V}^{\text{prev}}(\omega', a')\right)}_{\textbf{Interpolate} \text{ if } a' \notin \mathcal{G}_a}$$



Interpolation

Linear interpolation of $V^{\text{prev}}(\omega')$ at a'

1. Find m such that $a_m \leq a' \leq a_{m+1}$ (extrapolation: if $a' \geq a_{N_a}$ take $m = N_a - 1$)
2. Compute $t = \frac{a' - a_m}{a_{m+1} - a_m}$
3. Interpolate: $V^{\text{prev}}(\omega', a') \approx (1 - t) \cdot V^{\text{prev}}(\omega', a_m) + t \cdot V^{\text{prev}}(\omega', a_{m+1})$

- Linear interpolation preserves monotonicity and concavity/convexity.
- Can approximate non-linear functions if the grid is dense enough where the function has high curvature
- Interpolator is piecewise linear, continuous, but not differentiable at grid points

Note: You may use later other interpolation methods (e.g. spline). Have a look at Fatih Guvenen's lecture 2 for more details.

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

   (a) $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$ is independent of $a_j \Rightarrow$ For each ω_i, you can compute a vector $EV(a'|\omega_i)$

   Advantage: the maximand uses a single linear interpolation, cheap to evaluate!

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

   (a) $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$ is independent of $a_j \Rightarrow$ For each ω_i, you can compute a vector $EV(a'|\omega_i)$

      Advantage: the maximand uses a single linear interpolation, cheap to evaluate!

   (b) Compute $\mathbf{EV}(\omega_i, a')$ using matrix products: $\mathbf{EV} = \Pi \times \mathbf{V}^{\text{prev}}$

      Advantage: pre-compute only once per iteration on VF! Be very careful with dimensions if you have an additional state

      In QM1, we want you to code your own linear interpolation. But Matlab's `griddedInterpolant` on EV yields **high speedup**.

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

   (a) $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$ is independent of $a_j \Rightarrow$ For each ω_i, you can compute a vector $EV(a'|\omega_i)$

      Advantage: the maximand uses a single linear interpolation, cheap to evaluate!

   (b) Compute $\mathbf{EV}(\omega_i, a')$ using matrix products: $\mathbf{EV} = \Pi \times \mathbf{V}^{\text{prev}}$

      Advantage: pre-compute only once per iteration on VF! Be very careful with dimensions if you have an additional state

      In QM1, we want you to code your own linear interpolation. But Matlab's `griddedInterpolant` on EV yields **high speedup**.

2. In VFI, you need to evaluate the maximand many times (max), inside nested loops (over state space)

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

   (a) $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$ is independent of $a_j \Rightarrow$ For each ω_i, you can compute a vector $EV(a'|\omega_i)$

      Advantage: the maximand uses a single linear interpolation, cheap to evaluate!

   (b) Compute $\mathbf{EV}(\omega_i, a')$ using matrix products: $\mathbf{EV} = \Pi \times \mathbf{V}^{\text{prev}}$

      Advantage: pre-compute only once per iteration on VF! Be very careful with dimensions if you have an additional state

      In QM1, we want you to code your own linear interpolation. But Matlab's `griddedInterpolant` on EV yields **high speedup**.

2. In VFI, you need to evaluate the maximand many times (max), inside nested loops (over state space)

   (a) Use efficient optimizers (e.g. golden-section search) to evaluate as few times the maximands as possible

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

   (a) $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$ is independent of $a_j \Rightarrow$ For each ω_i, you can compute a vector $EV(a'|\omega_i)$

      Advantage: the maximand uses a single linear interpolation, cheap to evaluate!

   (b) Compute $\mathbf{EV}(\omega_i, a')$ using matrix products: $\mathbf{EV} = \Pi \times \mathbf{V}^{\text{prev}}$

      Advantage: pre-compute only once per iteration on VF! Be very careful with dimensions if you have an additional state

      In QM1, we want you to code your own linear interpolation. But Matlab's `griddedInterpolant` on EV yields **high speedup**.

2. In VFI, you need to evaluate the maximand many times (max), inside nested loops (over state space)

   (a) Use efficient optimizers (e.g. golden-section search) to evaluate as few times the maximands as possible

   (b) Remove nested loop: use a vectorized golden search on $a \in \mathcal{G}_a$

      MATLAB is faster when you operate on vector/matrices

How to code a fast *basic* VFI in MATLAB?

Remember: don't optimize prematurely! First get a working version, then make it faster.

1. At each iteration (on VF), pre-compute the expected *continuation* value $\mathbf{EV}(\omega, a') = \mathbb{E}_{\omega'|\omega} V^{\text{prev}}(\omega', a')$

   Baseline: inside the loop on $\mathcal{G}_\omega \times \mathcal{G}_a$, at each evaluation of maximand, compute $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$

   (a) $\sum_j \pi_{i,l} V^{\text{prev}}(\omega_l, a')$ is independent of $a_j \Rightarrow$ For each ω_i, you can compute a vector $EV(a'|\omega_i)$

      Advantage: the maximand uses a single linear interpolation, cheap to evaluate!

   (b) Compute $\mathbf{EV}(\omega_i, a')$ using matrix products: $\mathbf{EV} = \Pi \times \mathbf{V}^{\text{prev}}$

      Advantage: pre-compute only once per iteration on VF! Be very careful with dimensions if you have an additional state

      In QM1, we want you to code your own linear interpolation. But Matlab's `griddedInterpolant` on EV yields **high speedup**.

2. In VFI, you need to evaluate the maximand many times (max), inside nested loops (over state space)

   (a) Use efficient optimizers (e.g. golden-section search) to evaluate as few times the maximands as possible

   (b) Remove nested loop: use a vectorized golden search on $a \in \mathcal{G}_a$

      MATLAB is faster when you operate on vector/matrices

   (c) Parallelize the outer loop (over $\mathcal{G}_\omega$) using `parfor`. Not always faster! More in Jesus Fernandez Villaverde's slides.

Accuracy: Tolerance and precision levels

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

Tolerance level $\neq$ Precision level

Accuracy: Tolerance and precision levels

Tolerance level $\neq$ Precision level

- Tolerance level ε is an **input** $\rightarrow$ *When to stop iterating on the value function?*

  $\Rightarrow$ Typical values: $\varepsilon = 10^{-4}$ (low precision), 10^{-6} (medium), 10^{-8} (high)

  Note: The smaller ε, the longer the computation time (more iterations). But a small ε is not sufficient to guarantee a precise solution!

Accuracy: Tolerance and precision levels

Tolerance level $\neq$ Precision level

- Tolerance level ε is an **input** $\rightarrow$ *When to stop iterating on the value function?*

  $\Rightarrow$ Typical values: $\varepsilon = 10^{-4}$ (low precision), 10^{-6} (medium), 10^{-8} (high)

  Note: The smaller ε, the longer the computation time (more iterations). But a small ε is not sufficient to guarantee a precise solution!

- Precision level is an **output** $\rightarrow$ *How precise is my solution, given some properties of the true solution?*

  Usually: "The policy function should satisfy the Euler Equation (necessary & sufficient, if resource constraint holds)"

Accuracy: Tolerance and precision levels

Tolerance level $\neq$ Precision level

- Tolerance level ε is an **input** $\rightarrow$ *When to stop iterating on the value function?*

  $\Rightarrow$ Typical values: $\varepsilon = 10^{-4}$ (low precision), 10^{-6} (medium), 10^{-8} (high)

  Note: The smaller ε, the longer the computation time (more iterations). But a small ε is not sufficient to guarantee a precise solution!

- Precision level is an **output** $\rightarrow$ *How precise is my solution, given some properties of the true solution?*

  Usually: "The policy function should satisfy the Euler Equation (necessary & sufficient, if resource constraint holds)"

Checking Precision: Euler Errors

After convergence, compute the **Euler errors** at each point of the state space, using the policy functions

$$\mathrm{EE}(\omega_i, a_j) = \left| 1 - \frac{u'(c(\omega_i, a_j))}{\beta(1+r)\sum_l \pi_{i,l} \times u'\left(c\left(\omega_l, a'(\omega_i, a_j)\right)\right)} \right|$$

Additional references for this TD session

Main references:

- Heer & Maussner (2022), *DGE modelling*, 2nd edition, Chapters 4.1 and 4.2
- Azzimonti *et al.* (2025), *Macroeconomics*, Chapters 4.4 and 10.3, 10.4, 10.5 (link here)

Other references:

- Xin Yi's lecture notes on dynamic programming (link here)

  Nice starting point if you are lost
- Feodor Ishakov's lecture 40 on VFI (link here)

  Even has a youtube video explaining the process!
- QuantEcon's notebook on the stochastic growth model (link here)
- *[Advanced]* Fatih Guvenen's slides on dynamic programming and VFI (lectures 1, 2, 5: link here)