1 Notation

{sec:notation}

1.1 Tics, Tacs, Toes

The problem specified above assumes that the agent has only one decision problem to solve in any tic. In practice, it is increasingly common to model agents who have multiple choice tacs per tic; an agent's problem might have, say, a consumption decision (call it the c tac), a labor supply tac (call it ℓ) and a choice of what proportion ς of their assets to invest in a risky asset (the portfolio-choice tac).

The modeler might well want to explore whether the order in which the tacs are solved makes any difference, either to the substantive results or to aspects of the computational solution like speed and accuracy.

If, as in section **??**, we hard-wire into the solution code for each tac an assumption that its successor tac will be something in particular (say, the consumption tac assumes that the portfolio choice is next), then if we want to change the order of the tacs (say, labor supply after consumption, followed by portfolio choice), we will need to re-hard-wire each of the stages to know particular things about its new successor (for example, the specifics of the distribution of the rate of return on the risky asset would need to be known by whatever tac precedes the portfolio choice tac).

But one of the cardinal insights of Bellman's (1957, "Dynamic Programming") original work is that *everything that matters* for the solution to the current problem is encoded in a 'continuation-value function.'

Using Bellman's insight, we describe here a framework for isolating the tac problems within a tic from each other, and the tic from its successors in any future tic; the advantage of this is that the isolated tac and tic problems will then be 'modular': We can solve them in any order *without changing any code.* After considering the tac-order $[\ell, \mathrm{c}, \varsigma]$, the modeler can costlessly reorder the tacs to consider, say, the order $[\ell, \varsigma, \mathrm{c}]$.[1]

1.2 Toes

The key to the framework is distinguishing, within each tac's Bellman problem, three toes:

1. **Arrival**: Incoming state variables (e.g., k) are known, but any shocks associated with the period have not been realized and decision(s) have not yet been made

2. **Decision**: All exogenous variables (like income shocks, rate of return shocks, and predictable income growth $\mathcal{G}$) have been realized (so that, e.g., m's value is known) and the agent solves the optimization problem

3. **Continuation**: After all decisions have been made, their consequences are measured by evaluation of the continuing-value function at the values of the 'outgoing' state variables (sometimes called 'post-state' variables).

[1]As long as the beginning-of-tac and end-of-tac value functions for the tacs all depend on the same state variables; see the discussion in section **??** for further discussion.

When we want to refer to a specific toe in the tac we will do so by using an indicator which identifies that toe. Here we use the consumption tac problem described above to exemplify the usage:

Toe	Indicator	State	Usage	Explanation
Arrival	$\leftarrow$	k	$\mathrm{v}_{\leftarrow}(k)$	value at entry to tac (before shocks)
Decision(s)	(blank)	m	$\mathrm{v}(m)$	value of tac-decision (after shocks)
Continuation	$\rightarrow$	a	$\mathrm{v}_{\rightarrow}(a)$	value at exit (after decision)

Notice that the value functions at different toes of the tac have distinct state variables. Only k is known at the beginning of the period, and other variables take on their values with equations like $b = k\mathcal{R}$ and $m = b + \boldsymbol{\theta}$. We will refer to such within-the-tac creation of variables as evolutions.

1.3 Transitions

In the backward-induction world of Bellman solutions, to solve the problem of a particular tic we must start with an end-of-tic value function, which we designate by including the tic indicator in the subscript:

$$\mathrm{v}_{t\rightarrow}(a) \mapsto \beta \mathrm{v}_{\leftarrow(t+1)}(\overbrace{a}^{=k}), \tag{1}$$

{eq:trns-single-prd}

and we are not done solving the problem of the entire tic until we have constructed a beginning-of-tic value function $\mathrm{v}_{\leftarrow t}(k)$.

Once we are inside a tac, we will also need an end-of-tac value function. For the last tac in a tic the end-of-tac function is taken to be end-of-tic value function:

$$\mathrm{v}_{\rightarrow}(a) \mapsto \mathrm{v}_{t\rightarrow}(a). \tag{2}$$

One way to describe this is that when we are considering the solution to the current tac, we will be working with what, in computer programming, is called a 'local function' $\mathrm{v}_{\rightarrow}(a)$ whose value at the beginning of the tac-solution algorithm has been initialized to the value of a previously-computed 'global function' $\mathrm{v}_{t\rightarrow}(a)$ that had already been constructed by mapping itself to $\beta \mathrm{v}_{\leftarrow(t+1)}$ (equation (**??**)).

1.4 The Decision Problem in the New Notation

{subsec:decision-pro

The Decision problem can now be written much more cleanly than in equation (**??**):

$$\mathrm{v}(m) = \max_{c} \ \mathrm{u}(c) + \mathrm{v}_{\rightarrow}(\overbrace{m - \mathrm{c}}^{a}) \tag{3}$$

{eq:vMid}

whose first order condition with respect to c is

$$\mathrm{u}^{c}(c) = \mathrm{v}^{a}_{\rightarrow}(m - c) \tag{4}$$

{eq:upEqbetaOp}

which is mathematically equivalent to the usual Euler equation for consumption. (We will reuse this formulation when we turn to section **??**.)

Having defined these notational conventions, we are now ready to move to substance.