## 1 Notation

c:notation}

### 1.1 Periods, Stages, Perches

The problem so far assumes that the agent has only one decision. But agents often have multiple choices per period—for example, a consumption decision, a labor supply choice, and a choice of what proportion $\varsigma$ of capital $k$ to invest in a risky vehicle. We identify each stage type in two ways: by a **short-name** (a textual name, given when the stage is first introduced) and by a **control-name** (the stage's control variable, if any). For example, a labor supply stage might have short-name labor and control-name $\ell$; a consumption stage has control-name c. A stage list that constitutes a period may therefore be written by short-name, e.g. [labor, cons, disc], or by control-name, e.g. $[\ell, \mathrm{c}, \beta]$. A modeler might want to explore whether the order in which the stages are solved makes any difference, either to the substantive results or to aspects of the computational solution like speed and accuracy; with this scheme they do so merely by changing the order in which the stages are listed in the specification of the period.

If, as in section 2, we hard-wire into the solution code for each stage an assumption that its successor stage will be something in particular (say, the consumption stage assumes that the portfolio choice is next), then if we want to change the order of the stages (say, labor supply after consumption, followed by portfolio choice), we will need to re-hard-wire each of the stages to know new things about its new successor (for example, the specifics of the distribution of the rate of return on the risky asset must be known by whatever stage precedes the portfolio choice stage).

The cardinal insight 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 stage problems within a period from each other, and the period from its successors or predecessors in any other period. The advantage of this isolation is that each stage problem becomes a self-contained *module*: Its internal logic—the computation it performs on value functions—is defined independently of where it sits in the sequence of stages.

Modularity is valuable because it makes exploring such alternative model structures *cheap*. Using control-name indexing (e.g., the consumption stage by c), after considering the stage-order $[\ell, \mathrm{c}, \varsigma]$, the modeler can reorder the stages to consider, say, the order $[\ell, \varsigma, \mathrm{c}]$ *without rewriting any of the code that solves each individual stage.*[1] What must change are the *transitions*—the mappings that connect the end of one stage to the beginning of the next—which must be rewired to reflect the new ordering and its implied information structure. The stage-level code itself remains untouched.

[1]The beginning-of-stage and end-of-stage value functions for each stage must be defined over compatible state variables, so that the output of any stage can serve as the input to any other; see the discussion in section 8.

## 1.2 Perches

bsec:prchs}

The key is to distinguish, within each stage's Bellman problem, three viewpoints or 'perches' (we use that word to empasize that the perch does not *do* anything: It is merely a collection of mathematical and computational functions and objects).

1. **Arrival**: Incoming state variables (e.g., $k$) are known, but any shocks associated with the stage have not been realized and decision(s) have not yet been made
2. **Decision**: The agent solves the decision problem for the period
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)

This framework is silent about when shocks (if any) occur. In a consumption problem, the usual assumption is that shocks have been realized before the spending decision is made so that the consumer knows their resources when they decide how much to spend. But in a portfolio choice problem, the portfolio share decision must be made before the shock that determines the risky rate of return is known.

{perches}

**Table 1** Perches within the cons-with-shocks stage

| Perch | Indicator | State | value functions | Explanation |
|---:|:---:|:---:|:---:|:---|
| Arrival | $\prec$ | $k$ | $\mathrm{v}_{\prec} = \mathbb{E}_{\prec}[\mathrm{v}_{\sim}]$ | value at entry to stage |
| Decision(s) | $\sim$ | $m$ | $\mathrm{v}_{\sim} = \max_c \mathrm{u}(c) + \mathrm{v}_{\succ}$ | value of stage-decision |
| Continuation | $\succ$ | $a$ | $\mathrm{v}_{\succ}$ | value at exit |

This cons-with-shocks stage corresponds to the consumption problem defined above.

The table illustrates notation we can use when analyzing the problem from a context 'inside' a particular stage of a specific period. We require that no variable can have more than one meaning or interpretation inside a period, and we prohibit any reference to values of any variables or functions or other model objects from outside the stage (or period). This is why we use different letters, $k$ and $a$, to represent liquid resources before and after the consumption decision, even if ultimately this period's continuation value of $a$ will transmute into the next period's initial $k$.

In contrast, items like value functions $\mathrm{v}$ or expectations operators $\mathbb{E}$ have different meanings at different perches; we capture this using a subscript like $\prec$. The fact that all functions in a perch depend on the same state variables (shown in the second column) allows us to write those functions without specifying their arguments.

## 1.3 Builders and Connectors

rabsitdens}

Modularity requires that objects inside a period have no direct access to objects from any other period. This means that we must endow a stage or a period, at the time of its creation, with its end-of-stage-or-period value function $\mathrm{v}_{\succ}$.

For example, in a model in which every period contains only the single-stage consumption problem above, the continuation value function for the last (and only) stage at t will need to be 'connected' to the arrival value function in (t+1), which of necessity requires us to use t-related notation. Concretely, if we designate the end-of-*period* value function as $\mathrm{v}_{\succ(t)}$ (which is defined as the continuation value function from the last stage in the period), we use the notation

single-prd}

$$\mathrm{v}_{\succ(t)} := \beta \mathrm{v}_{\prec(t+1)}, \mathsf{U} \tag{1}$$

to describe what the builder does when constructing the predecessor to period $t+1$. The use of the ':=' signals creation: the left-hand side is *brought into existence* by the builder.

Tying two adjacent stages or periods together also requires that we define a Connector, $\mathcal{C}$, which specifies the relationship between the continuation-perch state variable(s) of the predecessor to the arrival-perch state variable(s) of the successor. Again concretely, for two successive periods each of which contains only a single consumption stage like the one described above, the Connector would look like:

-end-prd-v}

$$\mathcal{C}(a_t \leftrightarrow k_{t+1}). \tag{2}$$

## 1.4 Building the Pile Backward

lding-pile}

We call the structure in which accumulating periods are stored the pile `P`. Once backward induction is complete, the pile holds the full definition—what might colloquially be called "the full model." (We avoid the term "model" here because it is used in too many other ways and contexts.) Since we accrete the elements of `P` one by one as we iterate backward, it can be thought of as a pile of defined periods (and the Connectors between them).

The process of building the pile is straightforward. We start from the terminal period (section 3: $\mathrm{v}_T(m) = \mathrm{u}(m)$, consume everything), so initially $\mathtt{P} = \{\mathtt{p}_T\}$. We will denote the 'builder' as a computational object with notation like $\mathtt{B}^{\leftarrow}_{\mathrm{prd}}$, and we will speak of the backward-induction creation of a new period as being the result of 'applying' the $\mathtt{B}^{\leftarrow}_{\mathrm{prd}}$ to the existing `P`. The backward builder augments the `p` by creating a new Connector and then the new period's solution, so the pile then has the form $\mathtt{P} = \{\mathtt{p}_{T-1}, \mathtt{C}_{T-1,T}, \mathtt{p}_T\}$. Section 6 details the construction of $\mathtt{p}_{T-1}$; with multiple stages or control variables the structure generalizes as in section 8.