1 Notation

{sec:notation}

1.1 Periods, Stages, Steps

The problem so far assumes that the agent has only one decision problem to solve in any period. But it is increasingly common to model agents who have multiple choice stages per period; a problem might have, say, a consumption decision (call it the c stage), a labor supply stage (call it ℓ) and a choice of what proportion ς of their assets to invest in a risky asset (the portfolio-choice stage).

The modeler might well 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.

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 particular 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).

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 stage problems within a period from each other, and the period from its successors in any future period; the advantage of this is that the isolated stage and period problems will then be 'modular': We can solve them in any order *without changing any code* (only transitions need to be rewired). After considering the stage-order $[\ell, \mathrm{c}, \varsigma]$, the modeler can costlessly reorder the stages to consider, say, the order $[\ell, \varsigma, \mathrm{c}]$.[1]

1.2 Steps

The key is to distinguish, within each stage's Bellman problem, three steps:

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-stage and end-of-stage value functions for the stages all depend on the same state variables; see the discussion in section **??**.

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

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

Notice that the value functions at different steps of the stage have distinct state variables. Only k is known at the beginning of the stage, and other variables take on their values with equations like $b = k\mathcal{R}$ and $m = b + \theta$. We will refer to such within-the-stage creation of variables as 'evolutions.' So, the consumption stage problem has two evolutions: from k to m and from m to a.

```
# make empty model
# MNW, SB, AL to substitute their ideas here
# Using : as the python prompt

## create empty two-period-model object
>>> mdl2prd = HARK2.modelMake(modelKind=finiteHorizon,
                               [other configs])

## what does the solution object look like?
### - there is only one interval filled in, which is a blank slate:

>>> mdl2prd.solution

[
    interval
]

>>> prdT=0 # To make the stuff below a bit easier to follow
>>> mdl2prd.solution[prdT].interval

(returns whatever info is created for the default interval)

# Now let's build what is in the interval
## Assume there is a yml file 'vEndTerm' for constructing v(S)=0 for
    all states:

>>> mdl2prd.solution[prdT].interval.add_vEndPrd(
    parse_file_model='vEndTerm_plato.yml' # Platonic description
    parse_file_setup='mdl2prd_config.yml' # like, simulation_method=
        monte_carlo
    )

# probably we need to put the stages in a sublist ...
>>> mdl2prd.solution[prdT].interval.make_empty_stage_list()

# The last stage is added first
```

```
## Period T is a special case in which no solving is necessary since we
     know the solution is c=m

>>> mdl2prd.solution[prdT].interval.append_stage(
     stage_name='c',
     parse_file_model='consume_all_m.yml',
     parse_file_stage='consume_all_m_stagefile.yml'
  ) # not sure whether parse_file_stage is needed ...

# Now add a portfolio stage
## append_stage will test whether there are already any other stages in
     existence
## If so it will
### - prepend to the list of stages an empty earlier stage
### - set stg[new_stg].vEndStg = stg[successor_stg].vBegStg

>>> mdl2prd.solution[prdT].interval.append_stage(
     stage_name='Shr',
     parse_file_model='portfolio_shr_solve.yml',
     parse_file_model='portfolio_shr_stage.yml' # again not sure if
         needed
     )

# We are finished adding stages; finish by adding vBeg
>>> mdl2prd.solution[prdT].interval.add_vBeg() #
>>> mdl2prd.solution[prdT].interval.solution
     [vBeg, [ Shr, c ], vEnd]
```

1.3 Transitions

In the backward-induction world of Bellman solutions, to solve the problem of a particular period we must start with an end-of-period (continuation) value function, which we designate by explicitly including the period indicator in the subscript (the := symbol denotes that the object on the right hand side is assigned to the object on the left hand side):

$$\mathrm{v}_{t_\rightharpoondown}(a) := \beta \mathrm{v}_{\leftharpoondown(t+1)}(\overbrace{\ a\ }^{=k}), \tag{1}$$ {eq:trns-single-prd}

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

Similarly, in order to solve the problem of any stage, we must endow it with an end-of-stage continuation-value function. For the last stage in a period, the end-of-stage function is taken to be end-of-period value function; in our case where there is only one stage, this can be written cleanly as:

$$\mathrm{v}_{\rightharpoondown}(a) := \mathrm{v}_{t_\rightharpoondown}(a). \tag{2}$$

1.4 The Decision Problem in the New Notation

{subsec:decision-pro

From ‘inside’ the decision stage, the Decision problem can now be written much more cleanly than in equation (10):

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

{eq:vMid}