This document aims to outline the modifications we would need to have added to the configuration of models in Dolo yaml files in order for those config files to have the information necessary for them to be used for solving the kinds of models we currently solve in HARK.

1 The Problem - in Bellman Form

The usual analysis of dynamic stochastic programming problems packs a great many events (intertemporal choice, stochastic shocks, intertemporal returns, income growth, and more) into a small number of steps and variables. For the detailed analysis here, we will be careful to disarticulate everything that happens in the problem explicitly into separate steps so that each element can be scrutinized and understood in isolation.

We are interested in the behavior a consumer who begins period t with a certain amount of 'capital' $\mathbf{k}_t$, which is immediately rewarded by a return factor R_t with the proceeds deposited in a **b**ank account **b**alance:

$$\mathbf{b}_t = \mathbf{k}_t \mathsf{R}_t. \tag{1}$$

Simultaneously with the realization of the capital return, the consumer also receives noncapital income $\mathbf{y}_t$, which is determined by multiplying the consumer's 'permanent income' $\mathbf{p}_t$ by a transitory shock $\boldsymbol{\theta}_t$:

$$\mathbf{y}_t = \mathbf{p}_t \boldsymbol{\theta}_t \tag{2}$$

whose whose expectation is 1 (that is, before realization of the transitory shock, the consumer's expectation is that actual income will on average be equal to permanent income $\mathbf{p}_t$).

The combination of bank balances $\mathbf{b}$ and income $\mathbf{y}$ define's the consumer's 'market resources' (sometimes called 'cash-on-hand', following Deaton (1992)):

$$\mathbf{m}_t = \mathbf{b}_t + \mathbf{y}_t, \tag{3}$$

which are available to be spent on consumption $\mathbf{c}_t$.

The consumer's goal is to maximize discounted utility from consumption over the rest of a lifetime whose last period is date T:

$$\mathrm{v}_t(\mathbf{p}_t, \mathbf{m}_t) = \max_{\{\mathrm{c}\}_t^T} \quad \mathrm{u}(\mathbf{c}_t) + \mathbb{E}_t \left[\sum_{n=1}^{T-t} \beth^n \mathcal{L}_t^{t+n} \hat{\beta}_t^{t+n} \mathrm{u}(\mathbf{c}_{t+n}) \right] \tag{4}$$

$$\begin{aligned}
\mathbf{a}_t &= \mathbf{m}_t - \mathbf{c}_t \\
\mathbf{p}_{t+1} &= \mathcal{G}_{t+1} \mathbf{p}_t \Psi_{t+1} \\
\mathbf{y}_{t+1} &= \mathbf{p}_{t+1} \boldsymbol{\theta}_{t+1} \\
\mathbf{m}_{t+1} &= \mathsf{R} \mathbf{a}_t + \mathbf{y}_{t+1}
\end{aligned}$$

$$
\begin{aligned}
\mathcal{L}_t^{t+n}: &\quad \text{probability to } \mathcal{L}\text{ive until age } t+n \text{ given alive at age } t \\
\hat{\beta}_t^{t+n}: &\quad \text{age-varying discount factor between ages } t \text{ and } t+n \\
\Psi_t: &\quad \text{mean-one shock to permanent income} \\
\beth: &\quad \text{time-invariant 'pure' discount factor}
\end{aligned}
$$

Transitory and permanent shocks are distributed as follows:

$$
\begin{aligned}
\Xi_s &= \begin{cases} 0 & \text{with probability } \wp > 0 \\ \boldsymbol{\theta}_s/\wp & \text{with probability } (1-\wp), \text{ where } \log \boldsymbol{\theta}_s \sim \mathcal{N}(-\sigma^2_{\boldsymbol{\theta}}/2, \sigma^2_{\boldsymbol{\theta}}) \end{cases} \qquad (5) \\
\log \boldsymbol{\Psi}_s &\sim \mathcal{N}(-\sigma^2_{\boldsymbol{\Psi}}/2, \sigma^2_{\boldsymbol{\Psi}})
\end{aligned}
$$

where $\wp$ is the probability of unemployment (and unemployment shocks are turned off after retirement). The problem can be normalized by the level of permanent income $\mathbf{p}$, yielding the resulting Bellman representation:

$$
\begin{aligned}
\mathrm{v}_t(m_t) &= \max_{c_t} \quad \mathrm{u}(c_t) + \beth \mathcal{L}_{t+1} \hat{\beta}_{t+1} \, \mathbb{E}_t[(\boldsymbol{\Psi}_{t+1}\mathcal{G}_{t+1})^{1-\rho} \mathrm{v}_{t+1}(m_{t+1})] \\
&\text{s.t.} \\
a_t &= m_t - c_t \\
m_{t+1} &= a_t \underbrace{\left(\frac{\mathsf{R}}{\boldsymbol{\Psi}_{t+1}\mathcal{G}_{t+1}} \right)}_{\equiv \mathcal{R}_{t+1}} + \boldsymbol{\theta}_{t+1}
\end{aligned}
$$

Although our focus so far has been on the consumer's consumption problem once m is known, for articulating the steps of the computational solution it will be useful to distinguish a sequence of three steps within each time period (we use the word 'steps' to capture the proposition that we are not really thinking of these as occurring at different moments in time – only that we are ordering the things that happen within a given moment into an ordered sequence). The first step captures calculations that need to be performed before the shocks are realized, the middle step is after shocks have been realized but before the consumption decision has been made (this corresponds to the timing of v in the treatment above), and the final step captures the situation *after* the consumption decision has been made.

We need to define notation for these three steps. We will use t_- as a marker for the step in period t before shocks have been realized, and the t^+ as the indicator for the situation once the choice has been made, implicitly leaving the unmarked t to indicate the step in which the decision is made.

$$
\mathrm{v}_{t_-}(k_t) = \mathbb{E}_{t_-}[\mathrm{v}_t(\overbrace{k_t \mathcal{R}_t + \boldsymbol{\theta}_t}^{=m_t})] \qquad (6)
$$

$$
\mathrm{v}_t(m_t) = \mathrm{u}(\mathrm{c}_t(m_t)) + \mathrm{v}_{t^+}(a_t) \qquad (7)
$$

$$\mathrm{v}_{t^+}(a_t) = \beta \mathrm{v}_{t_-+1}(\underbrace{k_{t+1}}_{=a_t}) \tag{8}$$

If we turn off the mortality shocks ($\mathcal{L} = 1$) and family-size shocks, and redefine the pure time preference factor as $\beta = \beth$, the problem has first order condition

$$L\mathrm{u}'(c) = \beta \mathsf{R} \mathrm{v}'_{t^+}(m - c). \tag{9}$$

2 Creating An Instance of a Model

The first thing to do in building a model would probably be to create an empty model framework, perhaps with a command like:

```
mymodl = create_empty_model('modelsetup.yaml')
```

where any information that was globally true for the model (like, maybe, its name, some links to external resources describing it, etc) would be encapsulated in 'modelsetup.yaml').

The next thing to do would be to define at least one 'period' in the model, where a period describes a set of things that are envisioned to happen in a single time step:

```
mymodl = backward_create_a_period('periodsetup_terminal.yaml')
```

For simplicity, we will begin with a degenerate terminal period $T + 1$ (`Tp1`) in which the consumer arrives in the period with some amount of capital k but receives zero utility (because they are dead). (An alternative assumption would be to incorporate a bequest motive).

This period would consist of only a single 'stage' in which the beginning-of-period value function would require as an input the assets with which the consumer exited the prior stage, but the v(a) of those assets would be zero for every possible a:

```
periodsetup_Tp1.yaml:

symbols:
  statesBegOfStge: [a]

  values: |
    vFuncBegOfStge = 0.0        # zero value regardless of value of a
```

With that in place, we could create the last 'real' period. The first thing we might need to do is to define the transition between this new period and the one that already exists, which we might do with a file like `transitTm1.yaml`.

```
mymodl.backward_create_a_period('transitTm1.yaml')
```

transitTm1.yaml:

```
  transition:
    mymodl.solution[0].vFuncEndOfPrd = mymodl.solution[-1].stage[0].vFuncBegOfStge
```

which would mean that in the environment available to the model constructor in period $T-1$ the object `vFuncEndOfPrd` would be available (and would always evaluate to `vFuncEndOfPrd(a)=0`).

In practice, probably the right choice would be to automate this step, so that

```
mymodl.backward_create_a_period()
```

would do what we have defined in `transitTm1.yaml` by default. Only if the user wanted to do something different would it be necessary to have an explicit transition file.

Having defined this transition, the next step would be to specify other information that should be universally available during the entire period, which might be done with a file something like this:

periodsetup_normal.yaml:

```
symbols:
  statesBOP:  [k]             # statesEOP: [a] is implicit
  valuesBOP:  vFuncBegOfPrd
  parameters: [beta           # time pref
              ,rho            # RRA
              ,Lambda         # survival probability
              ,R              # riskfree return factor
              ,G              # permanent income growth factor
              ,sigma_ltheta   # std of log tran shock
              ,sigma_lpsi     # std of log perm shock
]

calibration:
  beta:         0.96
  G:            1.03
  rho:          2.0
  R:            1.04
  Lambda:       1.00
  sigma_ltheta:0.1
  sigma_lpsi:   0.1

options:
  grid: !Cartesian
      orders: [1000]
```

Next, one would construct at least one 'stage,' which consists of a set of steps that instantiate the structure of the model. The task of defining the solution would then be to define what happens serially backwards as each step gets added.

As with the between-period transition, we would presumably want to prepopulate the added stage with the assumption that the end-of-stage value function etc correspond to the beginning-of-stage value function for the following stage (in this case, the end of period value function).

So, for example, for the buffer stock consumption problem we would have a `coptstep.yaml` file that defines the consumption optimization step of a problem, which would have an `equations` block that might look something like this. (The assumption is that the step acquires its continuation value function and continuation states from the already-constructed prior stage).

`coptstep.yaml`:

```
symbols:
  statesBegOfStge: [m]
  controls:  [c]

  valuesBegOfStge: [vBegOfStge]
    value: |
      vBegOfStge = (c^(1-rho))/(1-rho) + beta*Lambda*vEndOfStge

    transition: |
      a = m - c

    arbitrage: |
      c^(-rho) - beta*Lambda*Rfree*vEndOfStge.deriv['a']

domain:
    m: [0.0, max_m]
```

where `vEndOfStge.deriv['a']` is a placeholder for whatever the correct notation would be to retrieve the derivative of `vEndOfStge` with respect to a and the assumption is that the equation defined as `arbitrage` presents an expression whose value should be zero at the optimum. The arbitrage equation therefore is equivalent to a rewritten version of our FOC (9):

$$\mathrm{u}^{c}(c) - \mathrm{v}^{a}_{t+}(m - c) = 0 \tag{10}$$

The result of processing this step should be the construction of the optimized value of c for each of the points in the state space defined as the input state space for each of the values of `m` constructed previously in the definition step, and construction of the `vMidOfStge` value function for the same states. (For the moment, we are not using EGM, because it is not clear how the syntax for EGM would work for the dolo).

We now need to define an earlier stage in the period, in which the stochastic shocks get realized, and expectations are constructed over the realizations of those shocks. Call this `shocks.yaml`.

```
equations:
  statesBegOfStge: [k]
  valuesBegOfStge: [vBegOfStge]


exogenous: !Normal
  Sigma: [[sigma_ltheta^2     , 0. ],
          [0.                 , sigma_lpsi^2]]
  Mu:    [-(sigma_ltheta^2)/2 ,(sigma_lpsh^2)/2]

  transition: |
      m = k*R/(G*exp(lpsi)) + exp(ltheta)

   expectorate: |
      vBegOfStge = vMidOfStge
```

where the expectorate operation would be defined to construct `vBegOfStge` by calculating the expectations at the grid of points for `k` defined in the `stagesetup.yaml` file.

3 What about Portfolio Choice? Depends on whether we want correlation with labor income

3.1 If rate of return uncorrelated with labor income shocks

Then we can define sequentially:
- add expectorate over tran and perm labor shocks given Rport and a - add optimize over alpha using usual expectations (not expectorations) - expectorate over k

3.2 If rate of return correlated with labor income shocks

- add expectorate over all three shocks given a - add optimize over alpha using usual expectations (not expectorations) - expectorate over k

3.3 Allow for both options simultaneously

Suppose you can choose to live in the correlated or uncorrelated world
Need to compute different values depending on which world you choose
Prior step would choose among the options

```
transition:
   a[stage-1] <-> k[stage]
   vEndOfStge[stage-1] <-> vBegOfStge[stage]
```

and possibly it would need to have further explicit information about things like how to map whatever grid exists for `k` in the current period to the corresponding grid for `a` in the prior period, and any information needed to map between the value functions.

With these files in place, the way to solve a period of the model might be something like this:

```
modl.add_period_preceding_existing_solutions = darkparse_period_setup('setupstage.y
modl.add_stage_to_current_period = darkparse_stagesetup('stagesetup.yaml')
modl.add_step_in_current_stage = darkparse('stagesetup.yaml')
modl.add_step_in_current_stage = darkparse('coptstep.yaml')
modl.add_step_in_current_stage = darkparse('expectorate.yaml')
modl.add_step_in_current_stage = darkparse('transit.yaml')
```

and at the end of this process the `modl` object would be ready to have another period added.

The payoff to all this work is that it should allow us to add components modularly. Suppose, for example, that we had the code to solve a portfolio choice problem of choosing the optimal share of risky versus safe assets; assuming that the risky return is realized at the same moment as the shocks to labor income (so that we can potentially allow the asset market return to be correlated with the labor supply decision – say, periods with bad stock marke performance are also periods when unemployment is higher), that decision problem should be insertable into the structure above between the `transit` step and the `expectorate` step:

```
modl.add_period_preceding_existing_solutions()
modl.add_stage_to_current_period()
modl.add_step_to_current_stage = darkparse('setup.yaml')
modl.add_step_to_current_stage = darkparse('coptstep.yaml')
modl.add_step_to_current_stage = darkparse('expectorate.yaml')
modl.add_step_to_current_stage = darkparse('portfoliostep.yaml')
modl.add_step_to_current_stage = darkparse('transit.yaml')
```

(This would also require specification of the stochastic process for the rate of return; following the scheme sketched above, this would presumably be located in the `stagesetup.yaml` file, though possibly we might decide that it makes more sense to describe the stochastic shocks in the `expectorate.yaml` file that calculates the expectations over those shocks).

4 Maybe there should be two kinds of transitions

Arguably, the transition between periods is such a different phenomenon than within-period transitions that the best way to deal with it might be to declare that each 'period'

is defined by a set of within-period steps, and a single transition equation that connects the period to its predecessor. In that case, the pseudocode for defining the solution to a period might look like this:

```
modl.add_period_preceding_existing_solutions()
modl.add_step_in_current_period = darkparse('setup.yaml')
modl.add_step_in_current_period = darkparse('coptstep.yaml')
modl.add_step_in_current_period = darkparse('expectorate.yaml')
modl.add_step_in_current_period = darkparse('portfoliostep.yaml')
```

followed by

```
modl.add_transition_to_prior_period = darktransparse('transit_to_prior_period.yaml'
```

.