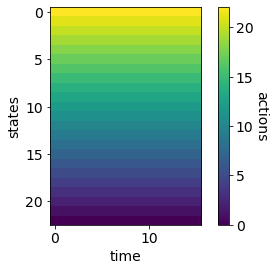
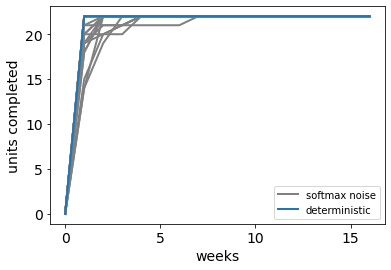
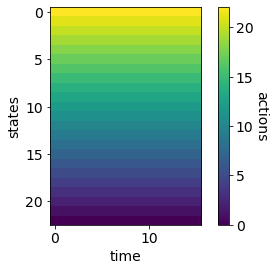
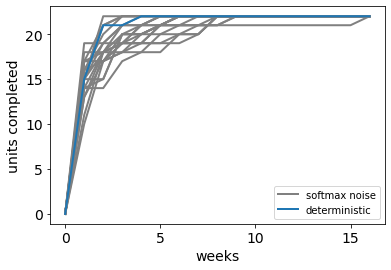
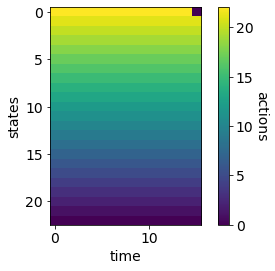
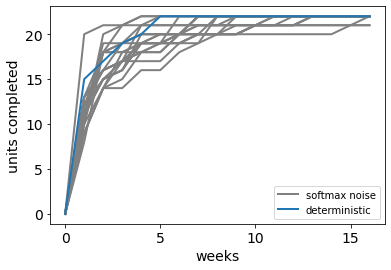
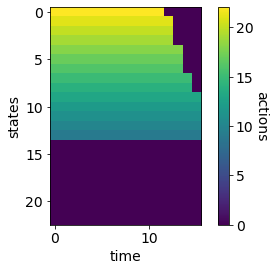
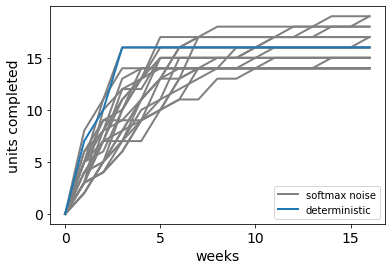
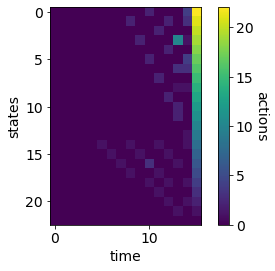
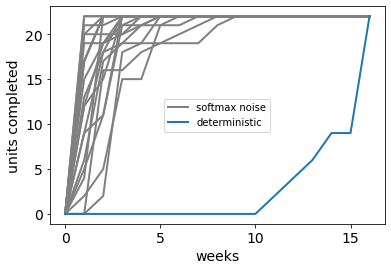
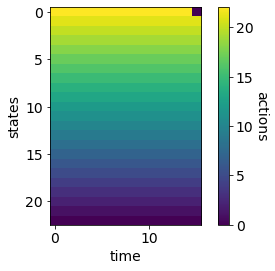
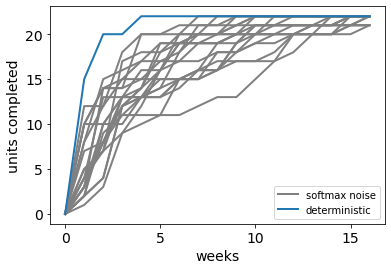
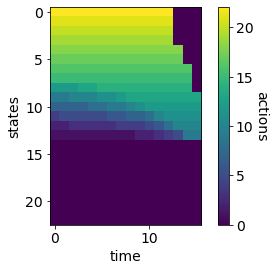
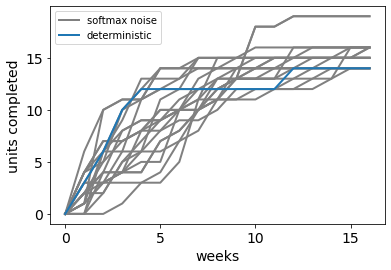
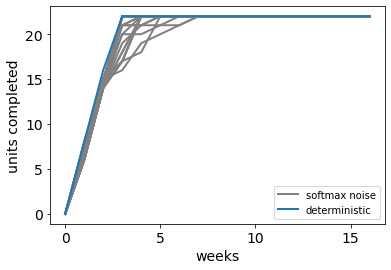
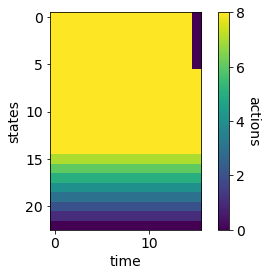
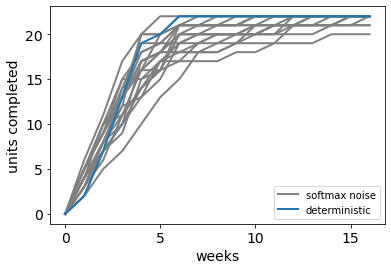
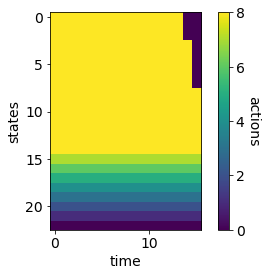
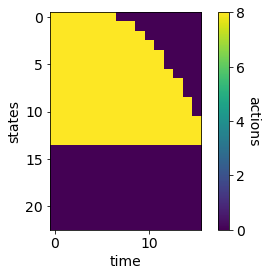
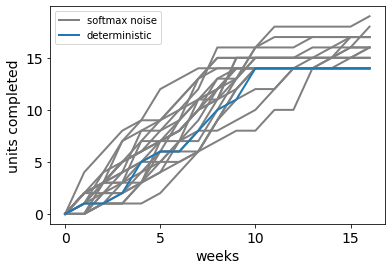
1. Without delay, single discount factor (everything else is the same as before):

Discount factor = 0.9  
Efficacy = 1.0

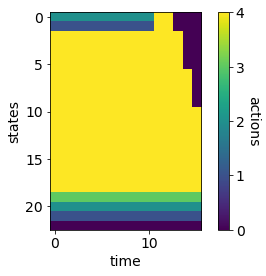
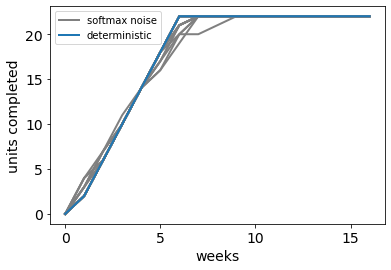
   
Efficacy = 0.7  
   
Efficacy = 0.5  
   
Efficacy = 0.2  
   
Since the rewards don’t come at the deadline anymore (but whenever the min number of credits are done), even with discounting, it is optimal to work immediately (and again as many as possible, to maximise chances of completion). For high efficacies, work max number possible. For lower efficacies, to reduce costs, instead of doing a few each week, try as many as possible until a point in time and states (after which don’t bother). As efficacies decrease, the progress curves get flatter. Note: with discount factor = 1.0, the progress curves get a bit flatter – because now it doesn’t matter whether you get reward now or later. So with efficacy = 1, everything is equally good until final week when work MUST be done if not completed yet. For other efficacies, policies look like discount factor < 1, but Q-values are still more similar across actions in the beginning. With discounting, there is a preference to get as much reward now than later, so more drive to finish earlier.   
  
discount factor = 1.0  
efficacy=1.0  
   
efficacy = 0.5  
 

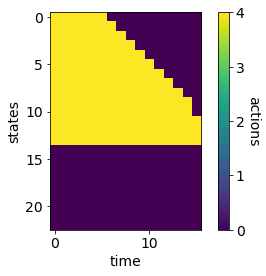
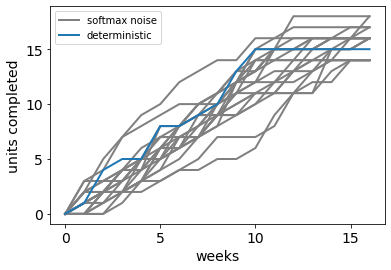
efficacy=0.2  
 

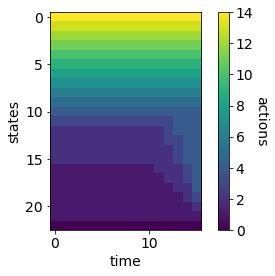
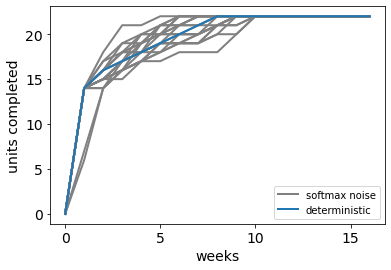
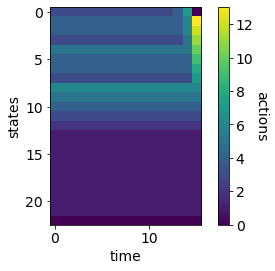
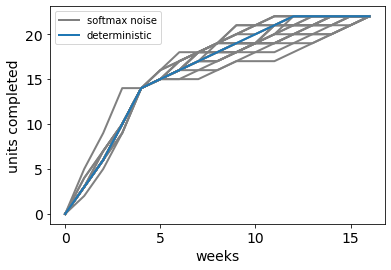
1. What if the number of units possible in a day are limited? So this would mean that 14 units cannot be reached immediately, then will there be procrastination?   
     
   Limiting to max 8 units per week (instead of allowing upto 22 per week). Even with this limit, with discount factor = 0.9, do as many as possible to expedite earning of credits (no reason to delay, in fact, just the opposite – preference to get rewards sooner than later due to discounting). Patterns of stopping similar to before  
     
   efficacy = 1.0  
     
    efficacy = 0.5  
     
   efficacy = 0.2

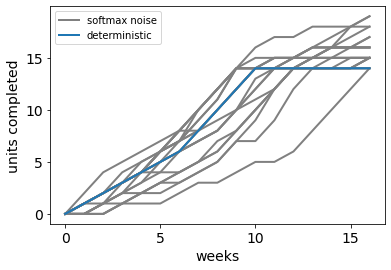
 

What if the limit is even lower?   
  
limit = 4 units per week; discount factor = 0.9   
  
Now, it is best to procrastinate a little bit (when number of units completed is low, because reward is still far away but efforts are immediate – at the same time, reward is self-paced, so the sooner you do, the sooner you get reward). Even with the limit = 8 case, decreasing discount factor introduces this type of procrastination. In general these initial delays are not perceptible (especially with noise)  
  
efficacy = 1.0;

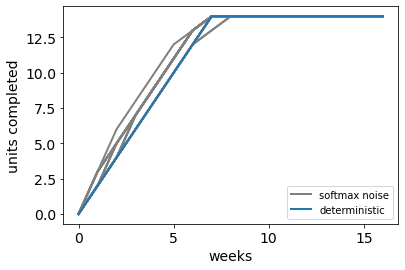
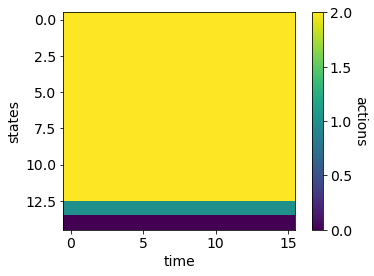
decreasing efficacy eliminates initial procrastination (coz you need to do earlier to ensure completion); for v low ones, still try in the beginning, but give up after a point; progress curves get flatter with decreasing efficacy  
efficacy = 0.3;   
 

1. What if we change the cost function such that every additional unit has a higher cost associated with it (so effort costs are convex in the number of units performed)  
     
   This ofc encourages fewer units per week – so higher the convexity, smaller the number of units tried per week. Beyond some level of convexity, the number of units are low enough such that it makes sense to put off earlier(do fewer at lower states when state=14 is still far away) and then more as higher states are reached.  
     
   effort cost = cost per unit \* (number of units)^exponent  
   exponent = 1.3  
      
   exponent = 2.0  
      
   The broader trends here are to still finish early, to get credits sooner. Not to delay till the end. Decreasing total rewards or discount factor makes progress slower (stopping after 14 credits are reached), increasing exponent makes the curves flatter (approaching straight line). But we see from data that people do delay and finish a lot at the end – how can we get it with this reward structure?

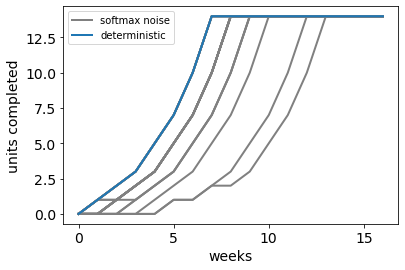
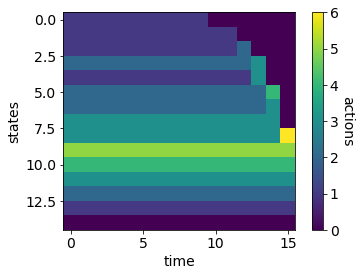
Exponent = 2.2, reward\_unit = 0.9

Some more about the ramping up shapes with and without convex costs and reward schedules:  
  
Peiyuan says in her thesis that with rational model she only gets ramping up till the end or steady working. This is shown averaged for all reward schedules. I agree for rewards with delay. But without delay, I don’t think this is the case. There are two types: with threshold (reward after 14 units) and immediate rewards.

As we showed before, without any limits or convexity in costs, it is best to finish immediately (for both schedules). With convexity, it is best to do as many as the convexity dictates for the immediate case; but for the threshold case, one should do fewer than the max one can do according to convexity – leading to a ramping up shape. But even then, it’s not ramping up till the end.

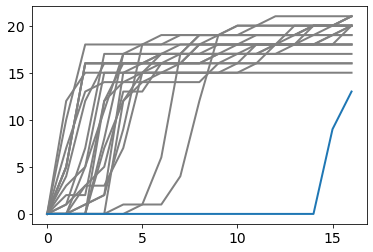
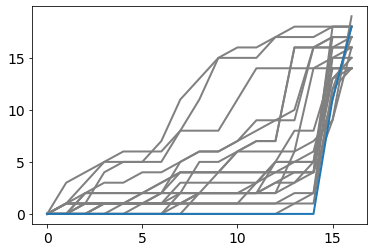
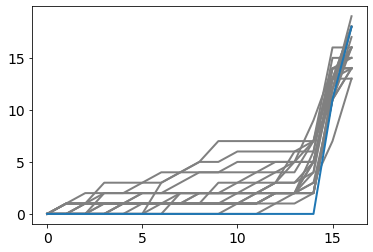
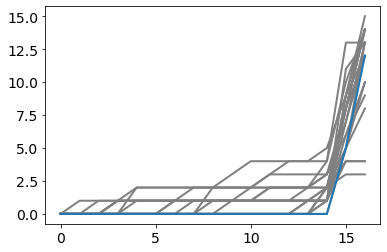
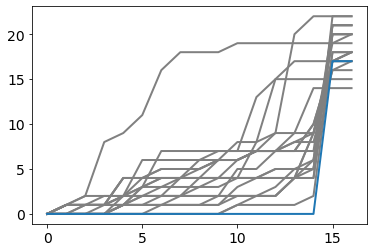
States = 14, discount\_factor = 0.6, reward\_unit = 4.3, reward\_shirk = 0.1, effort = -0.3, efficacy = 1.0  
convexity =   
  
immediate:  


Threshold:



1. Maybe having hyperbolic discounting causes people defect on their plans (of starting in time), and so work is put off till the end.  
     
   for ease, we set different discount factors for rewards and costs to simulate this.  
   for small enough rewards (relative to efficacy) and enough difference between discount factors, we get delays! However, only optimal to do till 14-ish credits, not more (due to low reward relative to efficacy)

Reward = 1.0 (everything else the same); no convexity in cost function  
efficacy = 0.6; discount\_factor\_reward = 0.9  
  
decreasing discount factor or increasing reward doesn’t improve delays (much, does happen a little bit in some param settings) but rather switches between delay or no delay at all. So gap between efficacies explains only small gaps. This is probably not the best explanation then (atleast for all patterns)? What about hyperbolic discounting?  
  
discount\_factor\_cost = 0.8 discount\_factor\_cost = 0.6

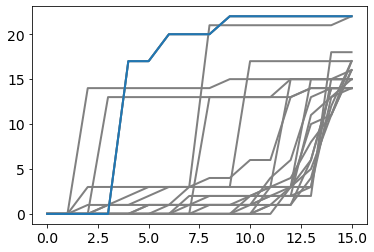
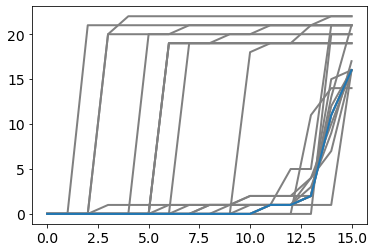
    
discount\_factor\_cost = 0.5  
  
 discount\_cost = 0.5, efficacy = 0.3 discount\_cost=0.5, efficacy = 0.3, assumed = 0.9  
 

1. What can other reasons be to delay?  
     
   They have other things to do, don’t know when that will get over. Without any info about what the other commitments are, we need to make some assumptions. But kinda like programming in the mechanism.  
     
    Another idea: what if the interest rewards from the units are variable (so some base rewards and interest rewards on top of that) So people must wait to see if they can get the higher ones. But just having probabilistic rewards is not enough (because then the policy will be calculated based on the expected value of the reward distributions). They must know the reward from doing a particular unit, so that they can wait for a higher one.   
     
   so we can have high reward and low reward super-states: we can progress in each of them (0-22 etc), there is some low probability of going to high reward states, so it could be optimal to wait.  
   In high reward states: some probability of getting high rewards for some units while in low reward state it’s all low rewards.

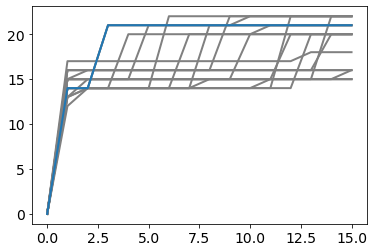
Reward\_unit = 2, reward\_interest = 2, reward\_extra = 0, Reward\_shirk = 0.1, effort\_work = -0.3  
p\_stay\_low = 1 – p\_stay\_h = 0.95

with sufficient interest rewards, it is best to wait for high reward state; increasing efficacy changes how quickly units can be completed

discount = 1, efficacy = 0.7 discount = 1, efficacy = 0.9

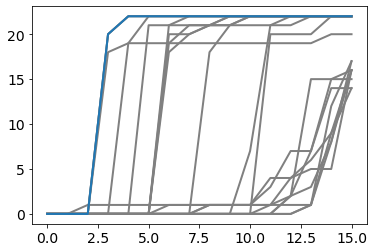
With low discount factors, it becomes more important to obtain immediate rewards, so quit waiting and seize whatever is available – so complete early  
Discount = .9, efficacy = 0.9



Changing statistics of interest reward availability  
when p\_stay\_h = p\_stay\_l, if H comes on, it stays for long

For high efficacy, not much difference between the two types of transitions (can do a lot in a single timestep)

Efficacy = 0.9, p\_stay\_h = p\_stay\_l = 0.95



For low efficacy:   
in switchy case, if H comes on some of the units are completed but H is switched away quickly, so it is completed later whenever H comes again or later towards deadline

In the other case, H stays for long in the trajectories where it comes on, so much quicker completion in them

Efficacy = 0.5, P\_stay\_l = p\_stay\_h = 0.95 Efficacy = 0.5, P\_stay\_l = 1-p\_stay\_h = 0.95  
