Appendix C.
Computing the optimal policy and information impulsivity

This appendix specifies how our model finds the optimal policy for the decision task described in appendix B. Appendix D provides the pseudo-code for this algorithm.

We vary the environment between agents. These environments are characterized by the mean and standard deviation of resource values ($\mu_{resource}$ and $\sigma_{resource}$, respectively), the mean and standard deviation of extrinsic event values ($\mu_{extrinsic}$ and $\sigma_{extrinsic}$, respectively), and the interruption rate (ρ). A policy π specifies what an agent should do in each state. The optimal policy, denoted by π^*, is the policy that maximizes fitness. We want to find the policy that maximizes an agents fitness, given its environment. From these policies we then compute the expected number of cues sampled, which is the inverse of information impulsivity.

Two observations. First, environmental conditions vary between agents, but are stable within an agent. Resources and extrinsic events are not depleted; observing a positive resource or extrinsic event does not reduce the expected resource quality or extrinsic event in future cycles. Rather, resources and extrinsic events are independent from cycle to cycle. The only change that 'carries over' from cycle to cycle is an agent's somatic state. As a result, the optimal policy has the Markov property: it is independent of previous cycle's outcomes, given the current somatic state. Second, there is an infinite number of future cycles. As long as the agent does not die from starvation there are infinite many future cycles.

Based on these two observations we conclude that the optimal policy does not change if we keep the somatic state at the start of a cycle, b_t, constant. We therefore can select a arbitrarily chosen cycle and compute the optimal policy for all possible somatic states. This set of policies is optimal for both that cycle and for all future cycles. Below we focus on this arbitrarily chosen cycle. For convenience, we refer to this cycle as the first cycle. We therefore compute, for all possible environments, the optimal policy for all possible somatic states $b_0 \in B$ that agent might have at the start of the first cycle.

In section 1 we first describe the decision the agent faces in terms of a Markov Decision Process. Specifically, we specify all possible actions, states, state transitions, and outcomes within a single cycle. In section 2 we describe how to find the optimal policy during the first cycle, assuming that the expected future outcome is known. We use stochastic dynamic programming to find the optimal policy. In section 3 we describe how to compute the expected future outcome of all subsequent cycles. We use value iteration to find these outcomes.

1. Specifying the MDP

1.1. Defining the state space. All possible states an agent can be in during an encounter can be described as a combination of its current somatic state b, the set of cues is has observed thus far D, and the cycle it is in at the moment t. Let S be the set of all possible states the agent can be in:

$$S = \{b, D, t | b \in B, D \in \overleftrightarrow{D}, t \in \mathbb{N}_0\} \tag{C.1}$$

Where $\overleftrightarrow{D}$ is a set-of-sets that contains all possible sets of cues the agent can observe. Let $|\cdot|$ denote the number of elements in a set. As an agent can sample up to 10 cues and the order of cues does not matter, $\overleftrightarrow{D}$ contains 66 combinations:

$$\left|\overleftrightarrow{D}\right| = \sum_{i=0}^{10} \frac{(2+i-1)!}{i! * (2-1)!} = 66\ combinations \tag{C.2}$$

Next we define four functions to retrieve properties of a state. First, let $b(s)$ be a function which results in an agent's somatic state in state s. Second, let $D(s)$ result in the set of all cues observed in state s. Third, let $t(s)$ result in that states cycle number. Fourth, let $s(b, D, t)$ refer to the state where an agent's somatic state is b, observed cues D, and is in cycle t.

An agent starts the first cycle with a somatic state b_0. Let S^{start} be the set of all possible starting states:

$$S^{start} = \{s(b_0, \emptyset, 0) | \, b_0 \in B\} \tag{C.3}$$

When an agent's somatic state is 0, it dies. We represent death with the special state s_{dead}:

$$s_{dead} = s(0,*,*) \tag{C.4}$$

Where $*$ can take any possible value.

1.2. Defining the action space. At each possible state $s \in S$ an agent can accept or reject the resource. If the maximum number of cues has not been reached in that cycle, an agent can also choose to sample. Let $A(s)$ be the set of all actions possible in state s:

$$A(s) = \begin{cases} \{accept, reject, sample\} & if\ |D| < 10 \\ \{accept, reject\} & otherwise \end{cases} \tag{C.5}$$

1.3. Transition functions. A transition function $P(s'|s, a)$ defines for each state s the probability of transitioning to any other state s' following action a. Every subsequent state s' that has a non-zero transition probability is called a successor state of s.

There is one absorbing state, s_{dead}, that when entered cannot be left:

$$P(s'|s = s_{dead},*) = \begin{cases} 1 & s' = s_{dead} \\ 0 & otherwise \end{cases} \tag{C.6}$$

1.3.1. Transition function when sampling. Should an agent sample, it's somatic state is first reduced by c, the cost of sampling. Next it moves to one of two successor states: one where it receives a positive cue, and one where it receives a negative one. Technically an agent can die by sampling a cue. Formally the transition function is:

$$P(s'|s, a = sample) = \begin{cases} 1 & b(s) \leq c \ \wedge s' = s_{dead} \\ Pr(c_+|D) & s' = s\big(b(s) - c, \{D(s), c_+\}, t(s)\big) \\ Pr(c_-|D) & s' = s(b(s) - c, \{D(s), c_-\}, t(s)) \\ 0 & otherwise \end{cases} \quad \text{(C.7)}$$

1.3.2. Transition function when accepting or rejecting. After accepting or rejecting, an agent enters a transition state. A transition state marks the end of the current cycle and, should the agent still be alive, the start of the next cycle. Because the transition states also denote the start of the next cycle they have no observed cues. Let $S^{transition}$ be the set of all possible transition state states, and let $s^{transition} \in S^{transition}$.

Should an agent accept, the resource encounter might be interrupted. Interruption occurs with probability ρ. If the resource is interrupted, the agent does not receive the resource quality $q \in Q$. If the somatic state is 0 after the resource encounter, the agent dies. If it lives, it will receive the extrinsic event $e \in E$ regardless of whether the encounter was interrupted. Should an agent reject, it only receives the extrinsic event $e \in E$. The agent dies if the somatic state reaches 0 after the extrinsic event. If the agent survives both the encounter and extrinsic event, it transitions to a transition state with its updated somatic state. To formalize, let $successorState(s, q, e)$ be a function that takes as input a current state, a resource encounter and an extrinsic event, and returns the resulting transition state:

$$\text{successorState(s, q, e)} \quad \text{(C.8)}$$
$$= \begin{cases} s_{dead} & s = s_{dead} \\ s_{dead} & b(s) + q < 0 \\ s_{dead} & b(s) + q + e < 0 \\ s^{transition}(b(s) + q + e, \emptyset, t(s) + 1) & otherwise \end{cases}$$

The transition probability of going from s to s' when accepting is then defined as:

$$P(s'|s, a = accept) \quad \text{(C.9)}$$
$$= \sum_{q \in Q} \sum_{e \in E} \begin{cases} \rho Pr(e) & s' = SuccessorState(s, 0, e) \\ (1 - \rho) Pr(q) Pr(e) & s' = SuccessorState(s, q, e) \\ 0 & otherwise \end{cases}$$

Should an agent decide to reject it only receives the extrinsic event:

$$P(s'|s, a = discard) = \sum_{e \in E} \begin{cases} Pr(e) & s' = SuccessorState(s, 0, e) \\ 0 & otherwise \end{cases} \quad \text{(C.10)}$$

2. Finding the optimal policy during the first cycle

To compute the optimal policy, we need to evaluate the expected fitness of each action in each state. To find these expected fitnesses we need to consider all possible successor states. Here we run into a problem. There are 66 non-transition states per encounter that an agent can reach when sampling. When an agent rejects there are $|E| = 201$ possible transition successor states. Likewise, should an agent accept there are there are $|E| = 201$ (when the encounter is interrupted) or $|E| * |V| = 40401$ (when there is no interruption) possible transition successor state. After this transition state the following cycle starts. To find the fitness outcome of the current state, we have to evaluate all possible states of all possible future cycles as well. Because there are an infinite number of cycles, this state space is infinitely large.

Fortunately, there is a tractable solution. First, assume that there is a function, $v_u^*(s^{transition})$ called the *optimal utility value function*. This function takes as argument a transition state reachable in the first cycle. It returns the (discounted) expected utility over all possible future cycles, given an agent's state at the start of the next cycle. Phrased differently, this function answers the question: "What expected discounted change in somatic state should an agent expect if it finishes the first cycle in this particular state?". We call this function the *optimal utility* value function because it stores the discounted expected outcomes (i.e., the utility) assuming that an agent follows the optimal policy from the next cycle onwards. We show how to compute $v_u^*(s^{transition})$ in section 3.

If we know $v_u^*(s^{transition})$, we can compute an agent's expected end-of-life somatic state after it completed the first cycle. Specifically, let S^{final} denote the set of all transition states reachable in the first cycle. Note that $S^{final} \in S^{transition}$. The expected end-of-life

somatic state when reaching state $s^{final} \in S^{final}$ is the somatic state of state s^{final} plus the utility of all future cycles. The utility of these future cycles is described in $v_u^*(s^{transition})$:

$$E\left[b_{t=\infty}\middle|s^{final}\right] = b(s^{final}) + \lambda * v_u^*(s^{final}) \quad \text{(C.11)}$$

Where $b_{t=\infty}$ is the end-of-life somatic state. We can use the expected end-of-life somatic state of a final state to compute the expected fitness of being in that state. These values are stored in the optimal fitness value function $v_f^*(s^{final})$:

$$v_f^*\left(s^{final}\right) = \omega(b(s^{final})) \quad \text{(C.12)}$$

2.1. Creating a decision tree: the forward pass. Now that the first cycle ends in final states, we can find the optimal policy for all possible starting states, using stochastic dynamic programming. First, using the action and transition functions, we can represent a cycle as a tree (i.e., a directed a-cyclical graph, see Figures C.1, C2, and C3). A tree is a collection of three sets: a set of state nodes, a set of action nodes and a set of edges.

A state node represents a state an agent can be in before making a decision. State nodes come in two flavors: transition state nodes that represent transition states (depicted as rectangles; red when rejecting and green when accepting) or non-transition state nodes (depicted here as large white circles). Each state node refers to exactly one state $s \in S$.

In each non-transition state an agent can accept or reject. Both actions result in that agent transitioning to a transition state with perfect certainty to the corresponding action node (black arrow). Alternatively, an agent might preform a sampling action. In that case it can transitions to one of two non-transition successor state (white circles), depending on whether sampling results in a positive or negative cue. We represent this stochasticity by letting an agent transition from the current state to an action node (a small black circle). From the action node an agent then transitions into a successor state. The transition function $P(s'|a, s)$ described above determines for each state-action pair the potential successor states, and how probable a transitions is. These

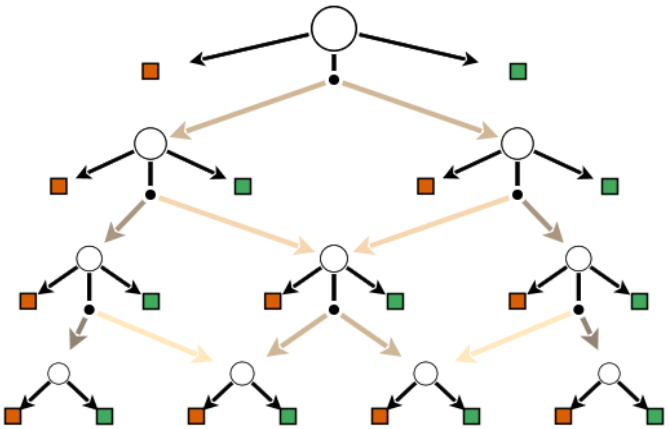

Fig C.1. An example decision tree after the forward pass. Non-transition states are depicted with large circles. In each non-transition state, an agent can reject (edge moving to the left), accept (edge moving to the right) or sample one more cue (middle edge). If sampling, there are two outcomes (small circle denotes an action state): an agent can receive a negative cue (moving left) or a positive one (moving right). The color of the edge shows the probability of positive or negative cues: more probable cues have a darker color. The radius of a state node depicts the probability of ending up in that state: large circles denote higher probabilities. There are many transition states for each action in each state (40401 for accepting, 201 for reject). Here we represent all transition states reachable by a single square.

transitions are represented using weighted directed edges (brown and yellow lines; darker colors indicate higher probabilities). These edges are weighted, representing the probability that an agent passes that edge after taking the action. Note that from some states an agent might transition to s_{death}. For ease of visualization, we include s_{death} in the transition states (the green and red rectangles).

The creation of this tree is called the *forward pass* of a tree. We create a tree for all possible starting states $s_{start} \in S^{start}$. After selecting a starting state, that state is *expanded*: for each possible action in that state an action node is added to the tree. Next, all possible successor states following that action are added to the tree. These successor states are subsequently expanded as well. This process of iteratively adding new states and expanded them continues until all possible states and actions are represented in the tree. Figure C.1 shows a truncated decision tree where an agent can sample up to a maximum of 3 cues. Note, however, that in our model 10 cues can be sampled. Representing this larger tree would be too unwieldy here. Moreover, since the branching factor is so large, all transition nodes accessible from a non-transition node are shown as a single rectangle, rather than as 40401 (when accepting) or 201 states (when rejecting).

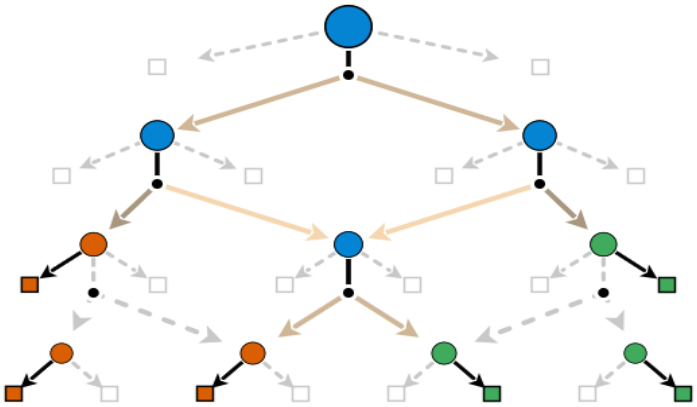

Fig C.2. An example decision tree after the backwards pass. Moving from the end of the tree (bottom) to the starting node (top), we compute for all non-transition states the best action. In green colored states the best action is to sample, in red colored states it is to reject, and in blue colored states it is best to sample. Notice that if an agent follows the optimal policy not all non-transition states are reachable. Here the bottom left and right most states are not reachable.

Although we can construct a tree for all cycles, we first construct a tree for the first cycle only. In section 3 we use this methodology for other cycles.

2.2. Finding the optimal policy: the backward pass. After the forward pass, we can compute the expected fitness of each state using stochastic dynamic programming and backwards induction. This is called the *backwards pass* (Figure C.2.)

Let $\omega\big(b(s)\big)$ be the expected fitness when an agent is in state s. Based on the previous section, we know that the fitness of a transition state in the first cycle is given by the optimal fitness value function:

$$\omega\left(b\left(s^{final}\right)\right) = v_f^*\left(s^{final}\right), \qquad s^{final} \in S^{final} \tag{C.13}$$

Using backwards induction, we can compute the expected fitness $\omega\big(b(s)\big)$ of being in a non-final state by computing the expected fitness at the 'bottom' of the tree (i.e., the states with the most cues sampled), and working backwards (i.e., towards the root node s_{start}). The fitness of being in a state s is the expected fitness of the fitness-maximizing action in that state. Note that multiple actions might be tied as the best action, in which case we assume that an agent randomly selects an action.

Formally, let $\omega\big(b(s|a)\big)$ be the expected fitness of performing an action a in state s, and let $A^*(s)$ be the set of all actions that maximize fitness. The expected fitness of being in a state s is then:

$$\omega\big(b(s)\big) = \begin{cases} v_f^*(s) & if\ s \in S^{final} \\ argmax_{a \in A(s)}\ \omega\big(b(s|a)\big) & otherwise \end{cases} \quad \text{(C.14)}$$

The expected fitness of an action a in state s is the sum of the fitness of each of successor states s', weighted for the transition probability to that successor state:

$$\omega\big(b(s|a)\big) = \sum_{s' \in S} P(s'|s,a) * \omega\big(b(s')\big) \quad \text{(C.15)}$$

The set of best actions for that state is then the set of all actions that result in the maximum fitness:

$$A^*(s) = \{a|\ \omega\big(b(s|a)\big) = \omega\big(b(s)\big)\} \quad \text{(C.16)}$$

2.3. Computing the number of cues sampled: the forward pruning pass. After calculating the fitness values of all states we can determine the optimal policy an agent should follow given that it starts the encounter in state $s_{start} \epsilon S^{start}$.

Let $S^{sampling}$ be the set of all non-transition states in which the set of best actions include the sampling action. Let S^{leaf} be the set of all non-transition states in which the sampling action is not in the set of best actions; after a leaf state the cycle ends in a transition state. Note that the states in S^{leaf} have no non-transition successors. Further note that since a tree is a-cyclical, an agent always visits precisely one leaf state. An agent might visit any number of sampling states before visiting a leaf state.

Suppose that an agent starts at the initial state an infinite number of times, each time following the same optimal policy. Because there is stochasticity in the transitions function (e.g., cues are stochastic), the same policy can result in different outcomes. Let $prop(s)$ be the proportion of an infinite population of runs in which an agent visits a particular state s_{start}. Because an agent will always visit precisely one leaf state the proportions of all leaf state sum to 1:

$$\sum_{l \in S^{leaf}} prop(l) = 1 \tag{C.17}$$

A state s is a *parent* of another state s' if there is at least one action where an agent might transition from s to. That is, if there is at least one edge connecting the states together via an action node. Let $parents(s)$ be a function that returns all parents of state s. An agent can only reach a state s if there is at least one parent state in which an agent samples. Thus, not all states in S^{leaf} are reachable. For instance, in Figure C.2 the bottom left and bottom right nodes are not reachable.

If the optimal policy in a parental state s contains only the action sampling, an agent will go to a successor state s' with a probability defined in the transition function $P(s'|s, a = sample)$ defined in section 1.3. However, if this parental state's set of best action includes another action besides sampling, an agent will sample in that state with a probability of $\frac{1}{|A^*(s)|}$, and perform a non-sampling action otherwise. We can extend this logic: let $\hat{P}(s'|s, a = sample)$ be the proportion of runs that an agent following the optimal policy transitions from s to s' after sampling, then:

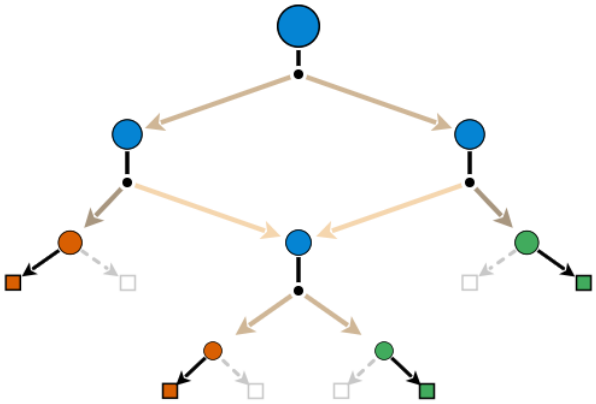

Fig. C.3. An example decision tree after the forward pruning pass. In the forward pruning pass all non-accessible state nodes are removed from the decision tree, and the probabilities of entering a state are renormalized. After renormalization the sum of the transition probabilities sums to 1 over successor states.

$$\hat{P}(s'|s, a = sample) = \begin{cases} 0 & if\ s \notin S^{sample} \\ \frac{1}{|A^*(s)|} * P(s'|s, a = sample) & otherwise \end{cases} \quad \text{(C.18)}$$

The proportion of a terminal state is equal to the transition function:

$$\hat{P}(s'|s, a = accept\ or\ reject) = \frac{1}{|A^*(s)|} * P(s'|s, a) \quad \text{(C.19)}$$

As all agents start out in s_{start} (i.e., the base case of this recursion), the proportion of that state is 1. We can therefore computer the proportion of any with the recursive function:

$$prop(s) \quad \text{(C.20)}$$
$$= \begin{cases} 1 & s = s_{start} \\ \sum_{parent \in parents(s)} \hat{P}(s|parent, A^*(parent)) * prop(parent) & otherwise \end{cases}$$

Figure C.3 shows the optimal policy after the forward pruning pass. The expected number of cues to sample under the optimal policy is then the number of cues sampled in each leaf node, weighted by the proportion of agents visiting that state:

$$E[\,cues\ sampled\,|\pi^*] = \sum_{l \in S^{leaf}} |D(l)| * prop(l) \tag{C.21}$$

3. Determining $V_u^*(s)$, the expected utility of future cycles

Previously we assumed that $v_u^*(s)$ was known. Here we describe how we compute $v_u^*(s)$ using Value Iteration (Sutton & Barto, 2018). There is no finite-time algorithm available to compute exact solution to this function. However, we can get an arbitrarily close approximation using epsilon-delta convergence.

$v_u^*(s)$ is the optimal utility value function for agents that go through an infinite number of future cycles. However, to compute this function we first assume that there are a finite number of k future cycles. A cycle's k number denotes how many cycles an agent will still have to go through, including the present one. Thus, in the $k = 1$ cycle an agent will only go through the final cycle – there is no additional cycle after the current one. In the $k = 0$ cycle an agent is finished: there are no more cycles.

If there are no more resource encounters or extrinsic events, the somatic state will not change anymore. Thus, the expected utility of that cycle is 0 for all possible starting states:

$$v_u^{k=0}(s_{start}) = 0 \tag{C.22}$$

Using the optimal utility value function for the $k = 0$ cycle, we can compute the optimal utility value function for the $k = 1$ cycle. We start with creating a decision tree for each possible starting state $s_{start} \in S^{start}$ for the $k = 1$ cycle. The construction of this tree can be done through the forward pass discussed above.

Next, we again use backwards induction by applying a backwards pass to all trees. This procedure is comparable as discussed above, with a few important difference. Most importantly, above we use backwards indication to find the *fitness* maximizing policy. Now we use backwards induction to find the expected *utility* of the optimal policy. First, we compute the expected immediate outcome of all transition states of all trees. The immediate outcome of a transition state is the difference between somatic state in that state and the somatic state of the state an agent started the cycle with:

$$O_{immediate}(s_{transition}|s_{start}) = b(s_{transition}) - b(s_{start}) \quad \text{(C.23)}$$

Next we compute the utility for all transition states in all trees. The utility of a transition state is the immediate outcome of that state, plus the utility of all subsequent cycles. The utility of all subsequent cycles is the utility of the $k = 0$ cycles, which is stored in $v_u^{k=0}(s^{terminal})$:

$$U(s_{transition}|s_{start}) \quad \text{(C.24)}$$
$$= O_{immediate}(s_{transition}|s_{start}) + \lambda * v_u^{k=0}(s_{transition})$$

Next, we can use backwards induction to find the utility of all non-transition states. We again start with the non-transition states at the end of the tree (i.e., the states with the most cues sampled), and working backwards towards s_{start}. For each node we compute the expected *utility* of taking an action a, for all possible actions. The expected utility of an action is the weighted sum of the expected utility of all possible successor states an agent can be in after taking that action:

$$U(s|a) = \sum_{s' \in S} P(s'|s,a) * U(s') \quad \text{(C.25)}$$

Because we assume that an agent behaves optimally in all states, the expected utility of a state is the expected utility of the best action:

$$U(s) = argmax_{a \in A(s)}(U(a|s)) \quad \text{(C.26)}$$

We continue going backwards through the tree until we find the expected outcome of the starting state s_{start}. The expected utility of a starting the $k = 1$ cycle in state s_{start} is then the expected utility of s_{start}. We store these expected discounted outcomes in $v_u^{k=1}(s)$:

$$v_u^{k=1}(s_{start}) = U(s_{start}), \qquad \forall s_{start} \in S_{start} \tag{C.27}$$

3.1. Iterating k and epsilon-delta convergence. After computing the expected utility of the final $k = 1$ cycle, we can compute the expected utility of the pen-ultimate $k = 2$ cycle. Specifically, we can again create decision trees for all possible starting somatic state in the penultimate cycle. After the forward pass we use backwards induction to find the expected utility of each starting state. However, rather than using the utility function $v_u^{k=0}(s_{start})$ in equation C.24 we use the newly computed $v_u^{k=1}(s_{start})$. After the backward pass we store the expected utilities for the penultimate cycle in $v_u^{k=2}(s_{start})$.

In general we can iteratively compute any cycles expected utility $v_u^{k=x}(s_{start})$ by replacing $v_u^{k=x-1}(s_{start})$ in equation C.24 with the utility function $v_u^{k=x-1}(s_{start})$. Theoretically we can find the optimal utility value function $v_u^*(s_{start})$ by iterating k an infinite number of times:

$$v_u^*(s_{start}) := v_u^{k=\infty}(s_{start}) \tag{C.28}$$

However, in practice this will take an infinite runtime. We can, however, get arbitrarily close using epsilon-delta convergence. Specifically, between two iterations we can calculate how large the absolute difference in expected utility is for two starting states with the same somatic states. If S_k^{start} denotes the set of all possible starting states in cycle *k*, then the delta is defined as the largest absolute difference between starting states with the same somatic state:

$$\delta(v_u^{k=x}, v_u^{k=x-1}) \tag{C.29}$$
$$= \max_{s \in S_x^{start}} \max_{r \in S_x^{start}} \begin{cases} abs\left(v_u^{k=x}(s) - v_u^{k=x-1}(r)\right) & if\ b(s) = b(r) \\ 0 & otherwise \end{cases}$$

A δ of 0 indicates that there is no change between two successive iterations. Because no other variables other than the optimal utility function differ over iterations, it follows that if $v_u^{k=x}$ and $v_u^{k=x-1}$ have a delta of 0, then so will $v_u^{k=x+1}$ and $v_u^{k=x}$. After his moment additional iterations will not change the optimal value function (i.e., the function has reached a stationary point).

$$v_u^{k=\infty}(s_{start}) = v_u^{k=x}(s_{start}) \quad if \quad \delta(v_u^{k=x}, v_u^{k=x-1}) = 0 \tag{C.30}$$

This process would take an infinite number of cycles. However, since δ decreases with increasing k, we can get δ arbitrarily close to 0. Hence, rather than iterating k until δ is 0, we stop when δ falls below some critical value ϵ.

$$v_u^{k=\infty}(s_{start}) \sim v_u^{k=x}(s_{start}) \quad if \quad \delta(v_u^{k=x}, v_u^{k=x-1}) < \epsilon \tag{C.31}$$

In our model we set ϵ to 0.001.

References

Sutton, R., & Barto, A. (2018). *Reinforcment Learning: An introduction*. *MIT Press* (2nd edition). Cambridge, Massachusetts: The MIT press. https://doi.org/https://doi.org/10.1109/tnn.1998.712192