Appendix C: An Annotated Example of the Matlab Code

C.1 Introduction

Here we provide an annotated example of the Matlab code—using the standard inversion and plotting routines available in SPM12—required to specify and solve a generative model. This reproduces the T-maze foraging example in chapter 7. This appendix, a little dry on its own, will be most useful for readers who attempt to implement this code in Matlab so that they can see it working. We recommend trying to "break" this demo by playing with different parameter values and changing the generative model. Only by doing this will an intuitive sense of the mechanics of Active Inference develop.

C.2 Preliminaries

We assume that readers have some familiarity with Matlab and have successfully downloaded the SPM12 software package from https://www.fil.ion.ucl.ac.uk/spm/. The first step is to ensure that the folder containing the SPM12 functions is added to the Matlab path. We then open a Matlab script and begin writing our demo by defining a function and giving it a name (here, `demo_AI_book`):

```
function demo_AI_book
rng default
```

Figure C.1

The second line in figure C.1 sets the random number generator to a default initial seed so that the same random numbers are generated each time the

function is used. We normally do this for demos, as this ensures reproducibility. However, this can be omitted if we are instead running this function multiple times to compute some summary statistics of behavior over multiple trials. It is good practice to include some comments here that tell people about the script. Given that this appendix is devoted to annotation of this script, we omit this documentation here.

Next, we define some of the important constants we will use later. The advantage of listing these together here is that it is easy for us to find them in case we want to perturb them later. We define two parameters that will play the role of probabilities, whose role will become clear in section C.3 (figure C.2):

```
a       = .98;
b       = 1-a;
```

Figure C.2

This definition ensures a + b = 1. Now we are ready to set up the **A**, **B**, **C**, and **D** matrices and vectors that define a generative model. In this simple simulation, we assume these parameters are the same in the generative model and the generative process.

C.3 The Likelihood

Our focus here is on how to formalize a likelihood matrix in Matlab, so we will not devote a great deal of space to describing the generative model or the paradigm it describes. (This description is given in section 7.2.) Our aim is to translate the likelihood matrices of figures 7.4 and 7.5 into a language that our inversion routines will understand. We start with $\mathbf{A}^1$, which is written as `A{1}`, where the term inside curly { } brackets corresponds to the superscript (i.e., outcome modality); see figure C.3. The elements of the matrix (or tensor, more technically) are addressable using three indexes. These are the outcome, the first (location) hidden state, and the second (context) hidden state. These appear inside normal ( ) brackets. The matrix for the first level of the second hidden state factor—the context in which the attractive stimulus is on the right—is then specified with a `1` index in

the third position. The context in which it is on the left is specified with a `2` index in this position:

```
A{1} (:,:,1) = [...
     1 0 0 0;     % start
     0 0 0 0;     % left cue
     0 1 0 0;     % right cue
     0 0 1 0      % left
     0 0 0 1];    % right
A{1} (:,:,2) = [...
     1 0 0 0;     % start
     0 1 0 0;     % left cue
     0 0 0 0;     % right cue
     0 0 1 0      % left
     0 0 0 1];    % right
```

Figure C.3

The rows are the outcomes, and the columns are the alternative levels of the first hidden state factor. Comparison with the matrices displayed in figures 7.4 and 7.5 should help clarify this syntax. The $\mathbf{A}^2$ matrices are similarly defined for context. Here, we make us of the `a` and `b` we defined at the top of the script (figure C.4):

```
A{2} (:,:,1) = [...
     1 1 0 0;     % reward neutral
     0 0 a b;     % reward positive
     0 0 b a];    % reward negative
A{2} (:,:,2) = [...
     1 1 0 0;     % reward neutral
     0 0 b a;     % reward positive
     0 0 a b];    % reward negative
```

Figure C.4

This completes our specification of **A**. We have probabilities defined for every combination of outcome (rows) in two modalities (superscript or curly bracket) for each combination of values for two hidden state factors (second and third indices).

C.4 Transition Probabilities

Following **A**, we now specify **B**. As discussed in chapter 7, the superscript associated with the **B** matrix refers to the hidden state factor, as opposed to the outcome modalities indicated by the superscript for **A**; again, we use curly brackets as equivalent to the superscript (figure C.5). Recall the two factors are location (`1`) and context (`2`). Each matrix maps from the state at the previous time (column) to the current time (row). The **B** matrices can vary with each action if the state factor is controllable. This means that control states require an additional index specifying which action is taken:

```
B{1}(:,:,1)    = [1 1 0 0; 0 0 0 0;0 0 1 0;0 0 0 1];
B{1}(:,:,2)    = [0 0 0 0; 1 1 0 0;0 0 1 0;0 0 0 1];
B{1}(:,:,3)    = [0 0 0 0; 0 0 0 0;1 1 1 0;0 0 0 1];
B{1}(:,:,4)    = [0 0 0 0; 0 0 0 0;0 0 1 0;1 1 0 1];

B{2}    = eye(2);
```

Figure C.5

Here we have specified the controllable location state with each of the four actions available. Recall that there are four available actions, each determining a transition to one of the four locations. The right and left arms are absorbing states, meaning the rat must stay in these once entered. The semicolons here indicate the end of each row of the matrix. The contextual states are not under the creature's control, so we do not need to specify a different set of transition probabilities for each action. We simply define a single identity matrix, denoted by `eye` in Matlab. This implements the specification in figure 7.6 and equation 7.5.

C.5 Prior Preferences and Initial States

Following **B** is **C**. Here we return to a similar syntax as **A**: both superscripts and curly brackets pertain to the outcome modalities. In addition, we specify **D** as we did **B**: superscripts and curly brackets relate to hidden state factors (figure C.6). The number of rows of **C** and **D** must correspond to the number of rows in the associated **A** and **B** matrices, respectively.

```
C{1}   = [-1 -1 -1;
           0  0  0;
           0  0  0;
           0  0  0;
           0  0  0];
c      = 6;
C{2}   = [ 0  0  0;
           c  c  c;
          -c -c -c];
D{1}  = [1 0 0 0]';
D{2}  = [1 1]'/2;
```

Figure C.6

The preferences are specified in the **C** matrices in terms of log probabilities (which do not need to be normalized). A difference of six between two outcomes means the more probable outcome is exp(6) times more likely. Each row pertains to a different outcome. Columns correspond to different time steps. This allows for the possibility of time-varying preferences. If only one column is specified, it will be assumed that preferences are the same at each time. The **D** vectors specified here simply ascribe a probability to each state for the start of the trial. See equations 7.6 and 7.7 for comparison. Note that in Matlab, a transpose is denoted by `'`.

C.6 The Policy Space

We specified the allowable actions implicitly through the **B** matrices. We could assume that the policies and actions are one and the same and that we select new policies every time step. Alternatively, we can specify policies as sequences of actions. One way of thinking about this is that each policy tells us which **B** matrix (indicated by the third index) is in play at each time step. We do this through specifying an array, `V` (figure C.7):

```
V(:,:,1) = [1 1 1 1 3 4 2 2 2 2
            1 2 3 4 3 4 1 2 3 4];
V(:,:,2) = 1;
```

Figure C.7

The first index of `V` (i.e., the rows) represents the position in the action sequence. This means the first row is the first action and the second row is the second action. As actions cause transitions, a three-step model only

requires two actions. The second index (i.e., the columns) represents the alternative policies that could be chosen. Finally, the third index is the hidden state factor. To aid intuition, `V(2,5,1) = 3` means that the fifth policy option involves selecting the location $\mathbf{B}^1$ matrix associated with the third action to transition from the second to the third time step. While in simple simulations one may include all possible policies, in other cases one may select a subset. Note, however, that the selection of the available policies is a design choice in itself and has implications for the resulting behavior.

C.7 Putting It Together

Having specified our POMDP, we now bring it all together in a single `mdp` variable (figure C.8):

```
mdp.V = V;              % allowable policies
mdp.A = A;              % observation model
mdp.B = B;              % transition probabilities
mdp.C = C;              % preferred outcomes
mdp.D = D;              % prior over initial states
mdp.S = [1 1]';         % true initial state
```

Figure C.8

The final line here lets us specify the true hidden states that we wish to start with. This information is not available to our simulated creature, who must infer states on the basis of the outcomes generated by these states and its generative model (specified in the first five lines).

C.8 Simulation and Plotting

The heavy lifting is completed behind the scenes by the `spm_MDP_VB_X` function, which implements the message passing and policy selection described in chapters 4 and 7. In addition, it simulates the world our creature must contend with, including transitions between states and the generation of outcomes. We could have included many other options; for details, we refer readers to the documentation for this function in the Matlab script.

Once we have simulated a trial, we often want to find some graphical representation of the results. Plots of the sort shown in figures 7.2 and 7.7 can be automatically generated with standard plotting routines:

```
MDP = spm_MDP_VB_X(mdp);
spm_figure('GetWin','Figure 1'); clf
spm_MDP_VB_trial(MDP);

spm_figure('GetWin','Figure 2'); clf
spm_MDP_VB_LFP(MDP,[],1);

spm_figure('GetWin','Figure 3'); clf
spm_MDP_VB_LFP(MDP,[],2);
```

Figure C.9

The lines of code in figure C.9 will simulate and plot the results in three figures. The first figure provides an overall summary of the simulation, including states, outcomes, policies selected, and retrospective inferences. The second figure shows the electrophysiological correlates of belief updating for the first hidden state factor; the third figure does the same for the second factor. We have reproduced the outputs of the first lines of code in graphical form (figure C.10) to reassure you that everything is working. We could have reproduced the other two; however, we wanted to give you the opportunity to engage in Active Inference and reduce your uncertainty about what the plots show by running the script yourself.

This concludes the simple annotated example, which we hope sets you on the path to exploring the range of generative models that can be specified using the same principles and syntax. You can find further examples by typing "DEM" into the Matlab command line and selecting demos from the resulting graphical user interface. The demo script we have provided is a simplified version of the routine used in Friston, FitzGerald et al. (2017). For further details, including learning of the generative model over multiple trials, we refer you to that paper and to the `DEM_demo_MDP_X.m` script available in SPM12.

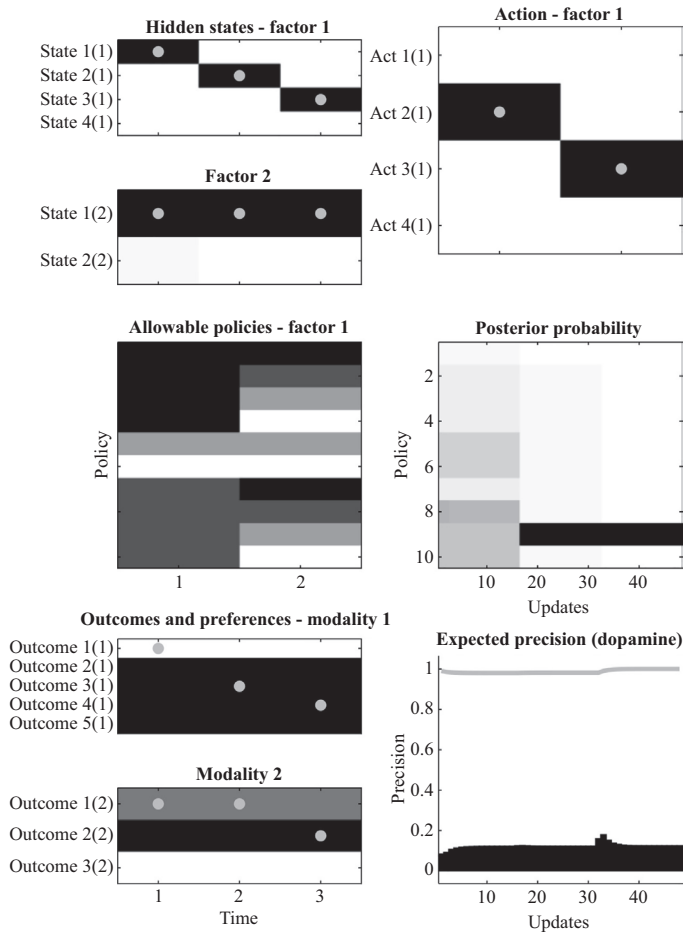

Figure C.10

Output obtained from the `spm_MDP_VB_trial` routine. *Upper left*: Beliefs held about each of the hidden state factors retrospectively (i.e., after all observations have been made). Black shading indicates a probability of one; white, of zero. Gray dots (which appear cyan when plotted in color)indicate the true states generated by the simulated environment; we see that the simulated rat accurately and confidently infers its location (Factor 1) and the context (Factor 2). *Middle left:* "Allowable policies" of the `V` variable specified in section C.6, showing a policy in each row: the first column is the first action taken; the second column, the second action. The different shades in each element represent different actions that could be chosen. *Lower left:* True outcomes (gray dots) in each modality. The background shading shows the C-matrices for each modality, with darker shades indicating preferred outcomes. *Upper right:* Inferred and selected actions. *Middle right:* Beliefs about each policy for each time step. *Bottom right:* Inferred precision of beliefs about the policies (gray line) with the rate of change plotted as a bar plot, reminiscent of the raster plots used to illustrate dopaminergic neuron firing in electrophysiology. Note that the time is specified in terms of updates; these refer to the iterations of a gradient descent on free energy. By default, there are sixteen updates per time step.