

## Appendix to Chapter 2

This appendix has two purposes. One is to reformulate the exemplar model using matrix algebra. This is important for speeding up the computational process, shortening the length of the code, and simplifying the program. Furthermore, matrix operators are quite useful for other topics covered in this book. The second purpose is to describe the computer program that was used to qualitatively analyze the predictions of the exemplar model (i.e., the program used to compute Figure 6.)

*Review of Matrix Algebra.* Matrix algebra is a mathematical formalism that based on objects called column vectors, row vectors, and matrices; as well as matrix operators such as transpose, matrix sum, scalar multiplication, inner product, matrix product, and Kronecker product, and matrix inverse. For our purposes, an  $n$  by  $m$  matrix is just a table of values that has  $n$  rows and  $m$  columns, and the entire table is denoted by a bold face letter such as  $\mathbf{X}$ . An example of a 2 (row) by 3 (column) matrix is shown below.

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 4 \end{bmatrix}.$$

The transpose of a matrix, symbolized by  $\mathbf{X}^T$ , simply means that we change the columns into rows.

$$\mathbf{X}^T = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 3 & 4 \end{bmatrix}$$

A column vector is just a matrix with only one column, such as for example

$$\mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}.$$

A row vector is just a matrix with one row, or it can be viewed as the transpose of a column vector

$$\mathbf{C}^T = [3 \quad 1 \quad -1].$$

It is often more convenient to use row vectors inside the text of a document.

Two matrices can be summed if they have the same dimensions, and the matrix sum is obtained by summing the corresponding elements. For example, if we use the  $\mathbf{X}$  matrix defined above and define another matrix as

$$\mathbf{D} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix} \text{ then } \mathbf{X} + \mathbf{D} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 4 \\ 4 & 2 & 6 \end{bmatrix}$$

Scalar multiplication means we multiply a single number times each element of a matrix:

$$2 \cdot \mathbf{X} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 0 & 2 \cdot 3 \\ 2 \cdot 2 & 2 \cdot 1 & 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 6 \\ 4 & 2 & 8 \end{bmatrix}.$$

An inner product is performed on row and column vectors of the same dimension, and this operation is denoted by  $\mathbf{C}^T \mathbf{B}$ . The inner product is defined as the sum of the cross-products of the coordinates for the two vectors. For example, using the definitions of  $\mathbf{C}^T$  and  $\mathbf{B}$  given above

$$\mathbf{C}^T \mathbf{B} = [3 \quad 1 \quad -1] \cdot \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = (3)(1) + (1)(0) + (-1)(2) = 1.$$

A matrix product between two matrices is defined by a matrix of inner products. If we multiply a  $n$  by  $m$  matrix  $\mathbf{X}$  times a  $m$  by  $p$  matrix  $\mathbf{Y}$  then the answer is a  $n$  by  $p$  matrix  $\mathbf{Z}$ . The element in the  $i$ -th row and  $j$ -th column of  $\mathbf{Z}$  is the inner product of the  $i$ -th row of  $\mathbf{X}$  with the  $j$ -th column of  $\mathbf{Y}$ . For example,

$$\mathbf{X} \cdot \mathbf{B} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} (1)(1) + (0)(0) + (3)(2) \\ (2)(1) + (1)(0) + (4)(2) \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \end{bmatrix}.$$

The Kronecker product between an  $n$  by  $m$  matrix and a  $p$  by  $q$  matrix is denoted  $\mathbf{X} \otimes \mathbf{B}$ , which forms a new matrix with  $n \cdot p$  by  $m \cdot q$  elements. Each element of the new matrix is formed by scalar multiplying each cell entry of the matrix  $\mathbf{X}$  by the entire matrix  $\mathbf{B}$ .

Consider, for example; the previous definitions of  $\mathbf{X}$  and  $\mathbf{B}$ :

$$\mathbf{X} \otimes \mathbf{B} = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 4 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{matrix} (1) \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \\ (2) \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \end{matrix} \quad \begin{matrix} (0) \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \\ (1) \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \end{matrix} \quad \begin{matrix} (3) \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \\ (4) \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \end{matrix} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 0 & 0 \\ 2 & 0 & 6 \\ 2 & 1 & 4 \\ 0 & 0 & 0 \\ 4 & 2 & 8 \end{bmatrix}.$$

We can define an element-wise function of a matrix as simply applying the function to each element of the matrix.<sup>1</sup> For example, the elementwise exponential function of the matrix  $\mathbf{B}$  is defined as

$$e^{\mathbf{B}} = \begin{bmatrix} e^1 \\ e^0 \\ e^2 \end{bmatrix}.$$

The elementwise square of the matrix  $\mathbf{B}$  is defined as

$$\mathbf{B}^2 = \begin{bmatrix} 1^2 \\ 0^2 \\ 2^2 \end{bmatrix}.$$

---

<sup>1</sup> It is important to note that there are ways to define a function of a matrix other than the one used here.

Finally, there are two special types of matrices that often appear in matrix calculations. One is a vector filled with all ones, denoted by  $\mathbf{J}$ . For example,

$$\mathbf{J} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \text{ This is useful for turning a scalar into a vector: } s \cdot \mathbf{J} = \begin{bmatrix} s \\ s \\ s \end{bmatrix}$$

or for summing a vector,  $\mathbf{J}^T \mathbf{B} = (1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2) = 3$ . Another matrix, called the identity matrix and denoted  $\mathbf{I}$ , has ones in the diagonal entries and zeros in all the off diagonal entries. For example

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

This matrix has the following special property:  $\mathbf{X} \cdot \mathbf{I} = \mathbf{X}$  in other words, it leaves  $\mathbf{X}$  unchanged.

The inverse of a matrix is used to perform matrix division. The inverse of the square matrix  $\mathbf{Y}$  is another square matrix which we denote as  $\mathbf{Y}^{-1}$ . The matrix inverse is defined by the property that  $\mathbf{Y}\mathbf{Y}^{-1} = \mathbf{I} = \mathbf{Y}^{-1}\mathbf{Y}$ . For example, if we define  $\mathbf{Y}$  as

$$\mathbf{Y} = \begin{bmatrix} 2 & 1 \\ 5 & 5 \end{bmatrix} \text{ then } \mathbf{Y}^{-1} = \begin{bmatrix} 1 & -.2 \\ -1 & .4 \end{bmatrix}, \text{ because } \begin{bmatrix} 2 & 1 \\ 5 & 5 \end{bmatrix} \cdot \begin{bmatrix} 1 & -.2 \\ -1 & .4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

As can be seen from this example, finding the inverse is not intuitively simple, but fortunately, inverse operators are available in mathematical programming languages. It can be used to perform matrix division as follows. Suppose  $\mathbf{Z}$  and  $\mathbf{Y}$  are known and we wish to solve for  $\mathbf{W}$  in the linear equation  $\mathbf{Y} = \mathbf{Z} \cdot \mathbf{W}$ , then the solution is

$$\mathbf{Z}^{-1} \mathbf{Y} = \mathbf{Z}^{-1} \mathbf{Z} \cdot \mathbf{W} = \mathbf{I} \cdot \mathbf{W} = \mathbf{W}.$$

*Exemplar model.* Now we shall use some of these concepts to rewrite the exemplar model in matrix terms. First we will compute all of the input activations for the exemplar model using matrix operators. Define  $\mathbf{X}$  as a column vector of ideal points. To be concrete, the stimuli range from 0 to 12 and we will use 13 equally spaced points defined by  $\mathbf{X}^T = [0 \ 1 \ 2 \ 3 \ \dots \ 11 \ 12]$ . The deviations between the ideal points and the stimulus values on each dimension can be represented by the two vectors

$$\mathbf{D}_1 = [\mathbf{X} - s_1(t) \cdot \mathbf{J}] / \sigma, \text{ and, } \mathbf{D}_2 = [\mathbf{X} - s_2(t) \cdot \mathbf{J}] / \sigma \quad (\text{A1a})$$

where  $\mathbf{J}$  is a 13 by 1 vector with all ones. We can compute the similarities for these deviations using elementwise matrix functions

$$\mathbf{Sim}_1 = e^{-\mathbf{D}_1^2}, \mathbf{Sim}_2 = e^{-\mathbf{D}_2^2}. \quad (\text{A1b})$$

Note that  $\mathbf{Sim}_1$  and  $\mathbf{Sim}_2$  are both 13 by 1 vectors. Then all of the elements contained in the square 13×13 grid of input nodes can be computed by the Kronecker product:

$$\mathbf{Sim} = \mathbf{Sim}_1 \otimes \mathbf{Sim}_2. \quad (\text{A2a})$$

$\mathbf{Sim}$  is a column vector with  $13^2$  similarities, and each element corresponds to Equation 6a of the previous chapter. The sum of all these similarities is obtained from the inner product  $\mathbf{J}^T \mathbf{Sim}$ , where  $\mathbf{J}$  is a  $13^2$  by 1 vector containing all ones. Finally the input activations for all of the input nodes contained in the  $13 \times 13$  grid are defined by

$$\mathbf{x} = \mathbf{Sim} / (\mathbf{J}^T \mathbf{Sim}). \quad (\text{A2b})$$

$\mathbf{x}$  is a column vector containing the  $13^2$  input activations, and each element corresponds to Equation 6b.

The connection weights connecting the two outputs to each of the  $13^2$  input nodes are represented as a 2 by  $13^2$  matrix denoted  $\mathbf{W}$ . When a stimulus is presented, the inputs are transformed into two output activations, one for each category, which is represented

by the row vector  $\mathbf{R}^T = [r_A, r_B]$  and  $\mathbf{R}$  is the column vector form for this. The matrix equation for the output activations is simply the matrix product

$$\mathbf{R}(t) = \mathbf{W}(t) \cdot \mathbf{x}(t) . \quad (\text{A3})$$

The row vector  $\mathbf{F}^T = [f_1(t), f_2(t)]$  represents the correct feedback on trial  $t$ , and  $\mathbf{F}$  is the column form for this. The matrix form for the delta learning rule is

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \cdot [\mathbf{F}(t) - \mathbf{R}(t)] \cdot \mathbf{x}(t). \quad (\text{A4})$$

This completes the matrix reformulation of the exemplar model.

These matrix formulas greatly speed up the computations for this model. In this example, we used only  $m = 13$  ideal points to detect stimulus values, which is a relatively small number. Suppose  $m = 121$  as it was in the example displayed in Figure 5. Then this increase in computational speed makes a huge difference. For example, rather than writing a series of  $m^2$  loops ( $13^2 = 169$  but  $121^2 = 14641$ ) to compute the outputs from all the input activations for each trial, only a single matrix operation is required. Matrix programming languages are extremely efficient at computing these matrix operations. In addition the code is simpler and easier to read and debug.

*Matlab computer program.* The *main* program is very simple. It simply sets some initial values, loads the stimuli, loops through all the decay rate and initial strength parameters on which we wish to perform the model tests, stores all these results in a large table called MT, and finally plot these values. The rows in the matrix MT represent different levels of decay rate, and the columns represent different levels of initial strength, and the cell values are 1 or 0 depending on whether the prediction from line 8 was successful or not, respectively. The function *spy* in Matlab is a function that plots a

point corresponding to each positive value of the matrix MT. This function generated Figure 6.

```

% main program

nr = 10; % (1) sets no. of rep's
load stimuli ; % (2) loads a file containing the
                stimuli in XAC XBC and P
aa = 25; AV = ((1:aa)./aa); % (3) set learning rate values
bb = 15; BV = (1:bb); % (4) set sensitivity values

MT=[]; % (5) Initializes test result matrix

sig = 5; % (6) Set discriminability

for i = 1:aa % (7) loop through predictions
    for j = 1:bb
        alf = AV(i) % (8) generate prediction
        b = BV(j) % (9) store result
        T = exemplar(alf,b,sig,XAC,XBC,P,nr,cutoff);
        MT(i,j) = T;
    end
end

spy(MT) % (10) plot results

```

The *exemplar* function is the heart of the program. First the initial values are set for variables. The constant, *ns*, represent the number of stimuli, which is 20 in this case. The line 2 finds the maximum stimulus value, which happens to be  $Max = 12$ . Then the program generates the 13 ideal points. The matlab code,  $0:Max$ , generates a row vector [1 2 ... 12]. Line 3 is used to construct a 2 by 13 initial weight matrix filled with zeros. The next statement just initializes two matrices that are later used to store results. Lines 4 – 10 perform the training for the model across all the stimuli and all the replications. Lines 6 and 8 generate the input and output activations by a function described later. The delta learning rule is applied in lines 7 and 9, according to Equation A4. Lines 11 – 15 perform the transfer test on the critical transfer stimuli. Lines 13 and 15 compute the

choice probabilities using Equation 4 from the previous chapter. The last line in this program performs the check to see whether the model correctly predicts the crossover interaction pattern.

```

function mT = exemplar(alf,b,sig,XAC,XBC,P,nr,cutoff)

ns = size(XAC,1); % (1) number of stimuli

Max = round(max(max(XAC))); % (2) defines the set of ideal points
I1 = 0:Max; I2 = 0:Max;

W = zeros(2,(Max+1)); % (3) Initial connection wghts

TA = []; TB = [];

for rep = 1:nr % (4) loop for replication
    for st = 1:ns % (5) loop for stimuli
        [In Out] = InOut(I1,I2,XAC,W,sig,st); % (6) Generate input and output
        WA = alf*([1 0]'-Out)*In'; % (7) delta rule
        [In Out] = InOut(I1,I2,XBC,W,sig,st); % (8) Generate input and output
        WB = alf*([0 1]'-Out)*In'; % (9) delta rule
        W = W + WA + WB; % (10) store wgt changes
    end
end

for st = 1:4 % (11) loop for transfer test
    if st == 2 | st == 3 % (12) test stim from cat A
        [In Out] = InOut(I1,I2,P,W,sig,st);
        pa = exp(b*Out); pa = pa(1)/(sum(pa)); % (13) prob choose A given cat A
        TA = [TA ; pa];
    else
        [In Out] = InOut(I1,I2,P,W,sig,st); % (14) test stim from cat B
        pb = exp(b*Out); pb = pb(1)/(sum(pb)); % (15) prob choose A given cat B
        TB = [TB ; pb];
    end
end

mT = ((TA(1)-TB(1)) > cutoff)*((TA(2) - TB(2)) > cutoff); % (16) Perform test of prediction

```

The function *InOut* performs two basic matrix operations. One is to construct the input node activations that we labeled **Sim** in Equation A2b above, and the other is to compute the output activations labeled **R** in Equation A3 above. Lines 1 – 2 perform the computations shown above as Equations A1a and A1b, and line 3 performs the

Kronecker product shown as Equation A2a and A2b. The last line computes the output using Equation A3.

```
function [In , Out] = InOut(I1,I2,X,W,sig,st)
in1 = 1./exp(((I1-X(st,1))/sig).^2);           % (1) Sim for s1
in2 = 1./exp(((I2-X(st,2))/sig).^2);           % (2) Sim for s2
In = kron(in1,in2); In = (In./sum(In))';       % (3) Grid of input activations
Out = W*In;                                     % (4) Input - Output Activations
```

This program could have been written just as easily using another programming language such as Mathematica, Gauss, or SAS.