Cerebellum and Motor Control

Part 1

1. **Run the 1-state model: The 1-state model is set up to run subject 1 however is currently missing the B (learning) component (line 28) based on:**

   $Z^{(n+1)} = A\, z^n + B\,(D.u^n - z^n)$

   Complete line 28 and submit this line of code.

   Here's the modified code after adding the B component

```
z(n+1, D.targetnum(n)) =
M.A*z(n,D.targetnum(n))+M.B*(D.u(n)-z(n,D.targetnum(n)));
```

2. **Complete the 2-state model: The 2-state function ('fits_all_ts') has a line missing; line 69. This relates to the updating of the slow state (zs). Based on:**

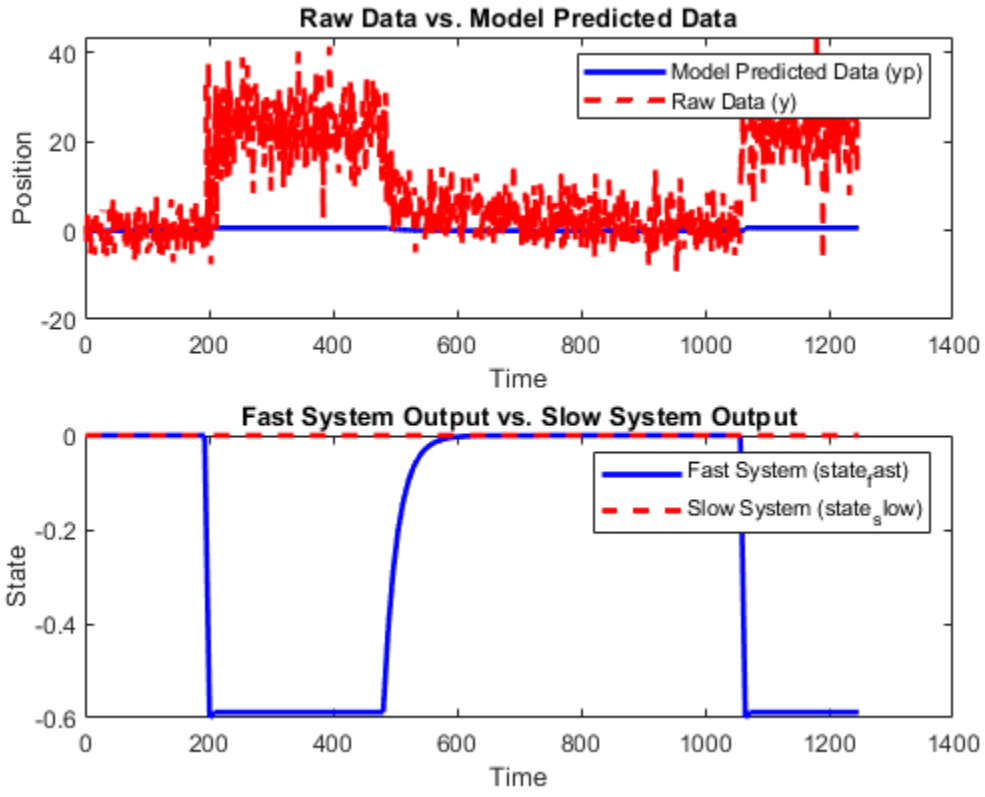
$$x_1(n+1) = A_f \cdot x_1(n) + B_f \cdot e(n)$$
$$x_2(n+1) = A_s \cdot x_2(n) + B_s \cdot e(n)$$

   and how the fast state is updated on line 68, complete this line. Submit this line of code.

   Here's the modified code:

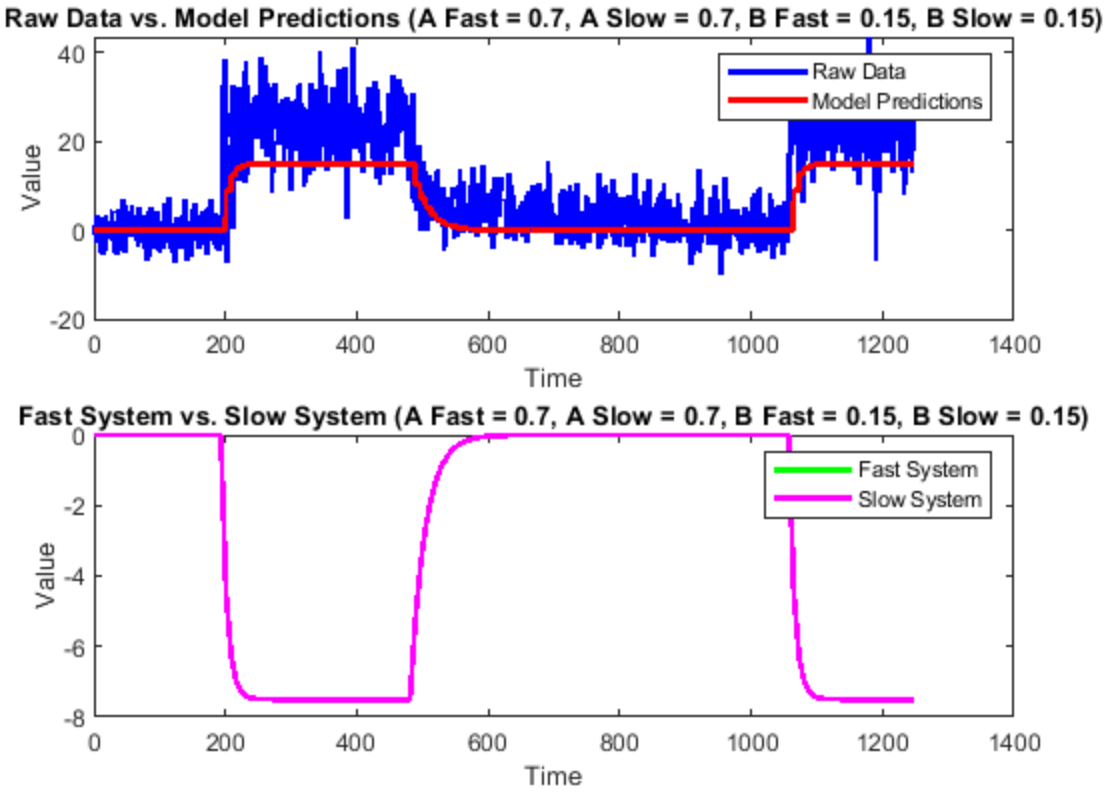
```
zs(n+1, D.targetnum(n)) = M.A(2) * zs(n, D.targetnum(n)) + M.B(2) *
(D.u(n) - zf(n, D.targetnum(n))-zs(n, D.targetnum(n)));
```

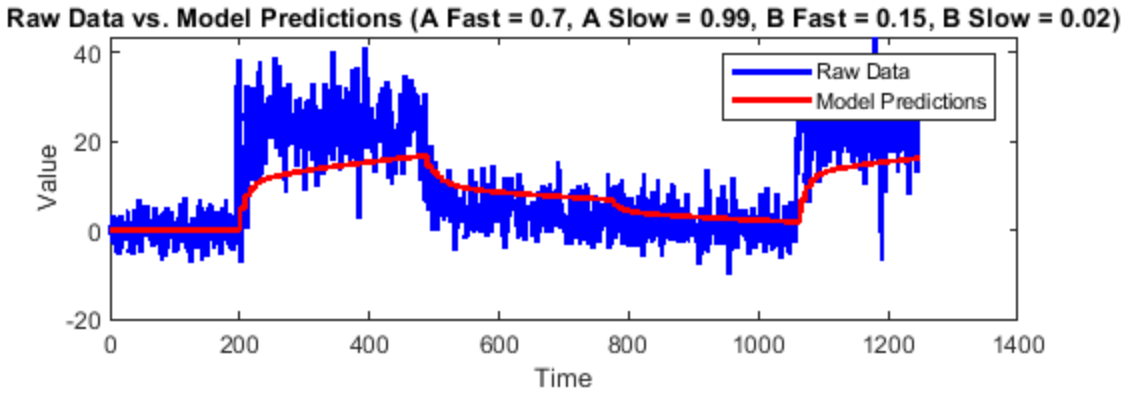
3. **Run the 2-state model: The model is set to run subject 1. Run the model. Plot the raw data (y) against the model predicted data (yp) and the output of the fast system (state_fast) and slow system (state_slow). Submit this figure.**

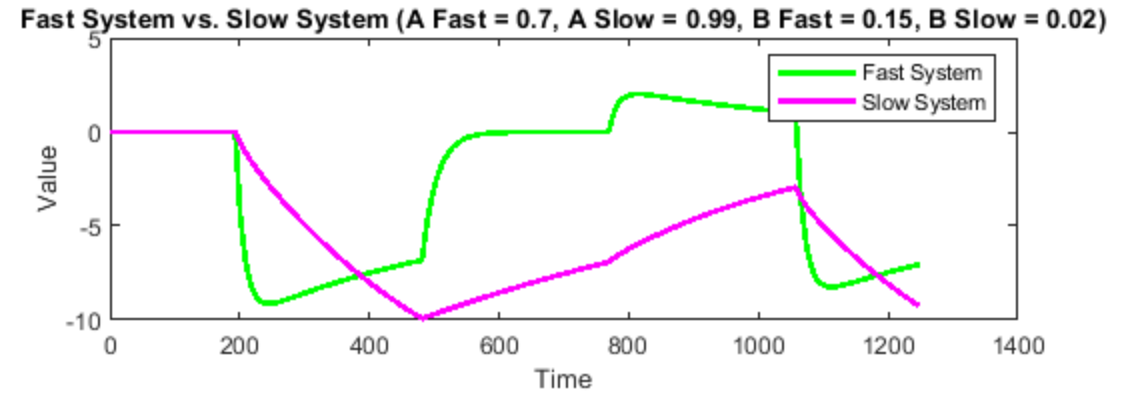


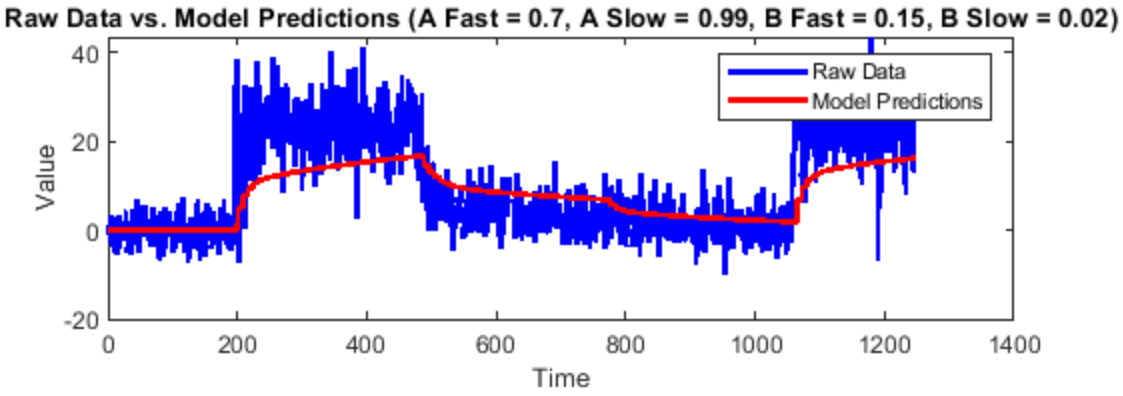
Part 2

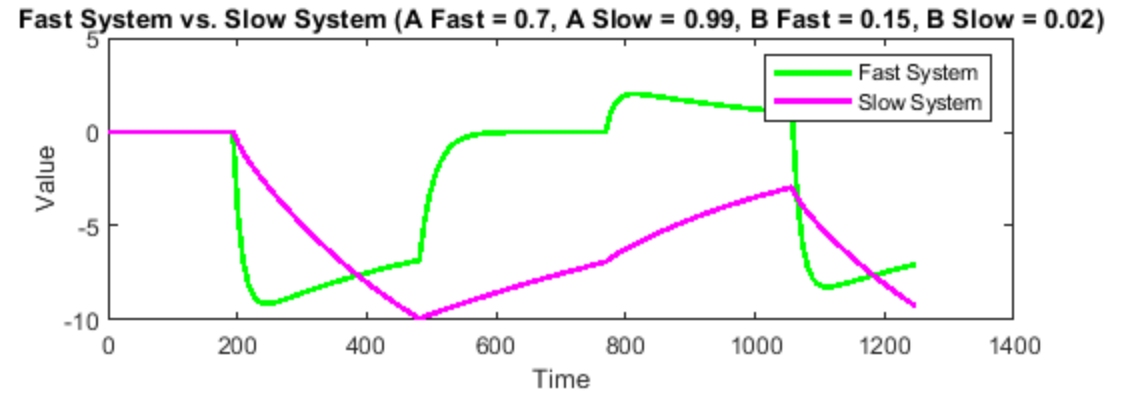
4. **Next, change the A (retention) and B (learning) values in the model for the both the fast and slow system ('fit_all_ts'). Do this one at a time, remembering to return the other values back to their original number. Plot these 4 predictions (yp) against the raw data. Submit this figure.**

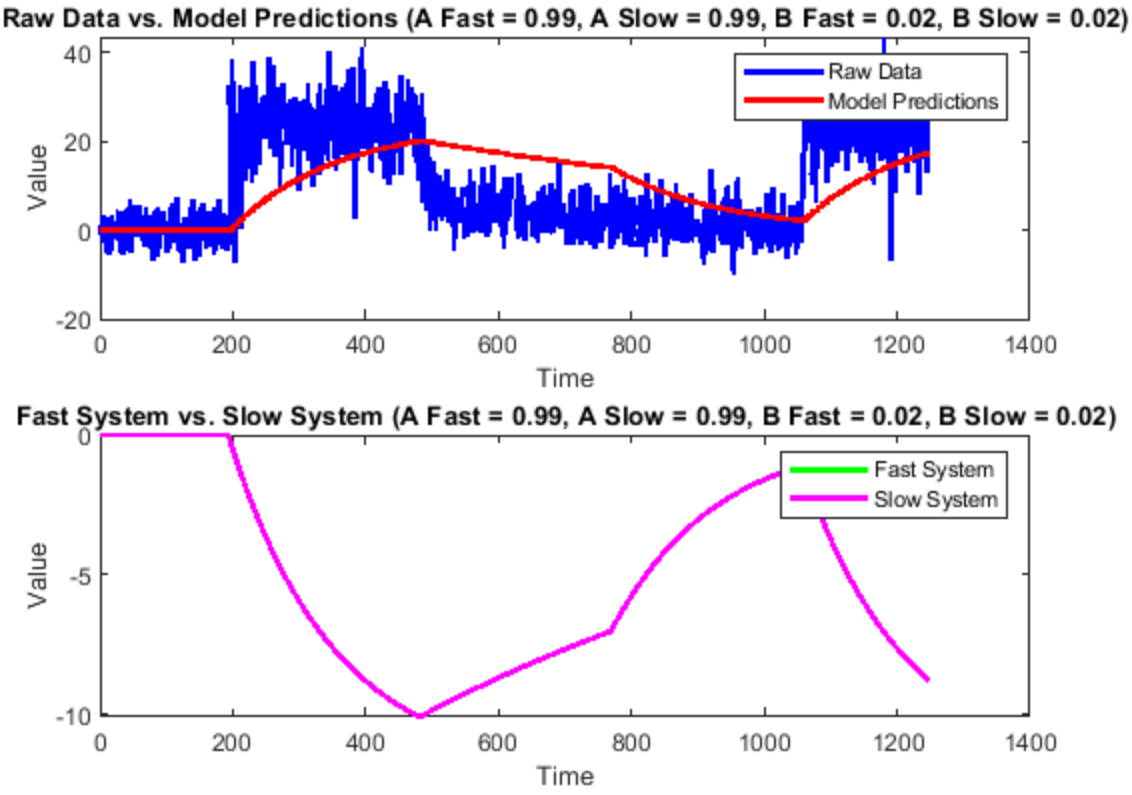

Raw Data vs. Model Predictions (A Fast = 0.7, A Slow = 0.7, B Fast = 0.15, B Slow = 0.15)
Raw Data
Model Predictions
Value
Time
40
20
0
-20
0
200
400
600
800
1000
1200
1400
Fast System vs. Slow System (A Fast = 0.7, A Slow = 0.7, B Fast = 0.15, B Slow = 0.15)
Fast System
Slow System
Value
Time
0
-2
-4
-6
-8


Raw Data vs. Model Predictions (A Fast = 0.7, A Slow = 0.99, B Fast = 0.15, B Slow = 0.02)
Raw Data
Model Predictions
Value
Time
40
20
0
-20
0
200
400
600
800
1000
1200
1400


Fast System vs. Slow System (A Fast = 0.7, A Slow = 0.99, B Fast = 0.15, B Slow = 0.02)
Fast System
Slow System
Value
Time
5
0
-5
-10
0
200
400
600
800
1000
1200
1400


Raw Data vs. Model Predictions (A Fast = 0.7, A Slow = 0.99, B Fast = 0.15, B Slow = 0.02)
Raw Data
Model Predictions
Value
Time
40
20
0
-20
0
200
400
600
800
1000
1200
1400


Fast System vs. Slow System (A Fast = 0.7, A Slow = 0.99, B Fast = 0.15, B Slow = 0.02)
Fast System
Slow System
Value
Time
5
0
-5
-10
0
200
400
600
800
1000
1200
1400

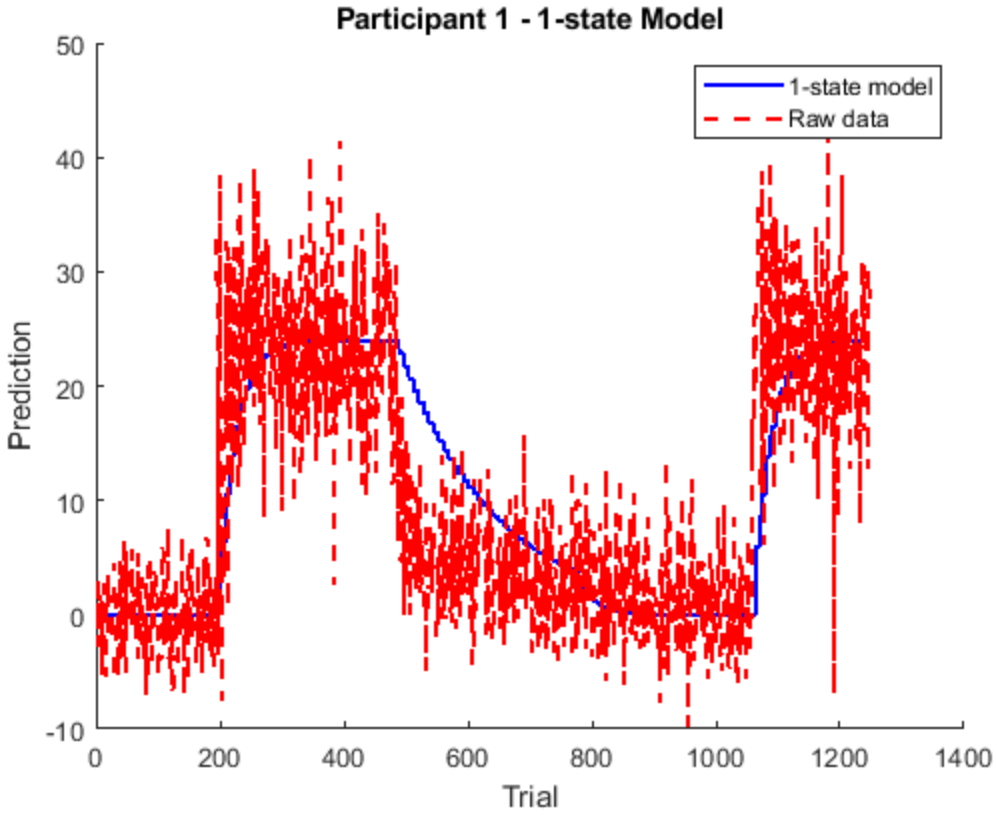


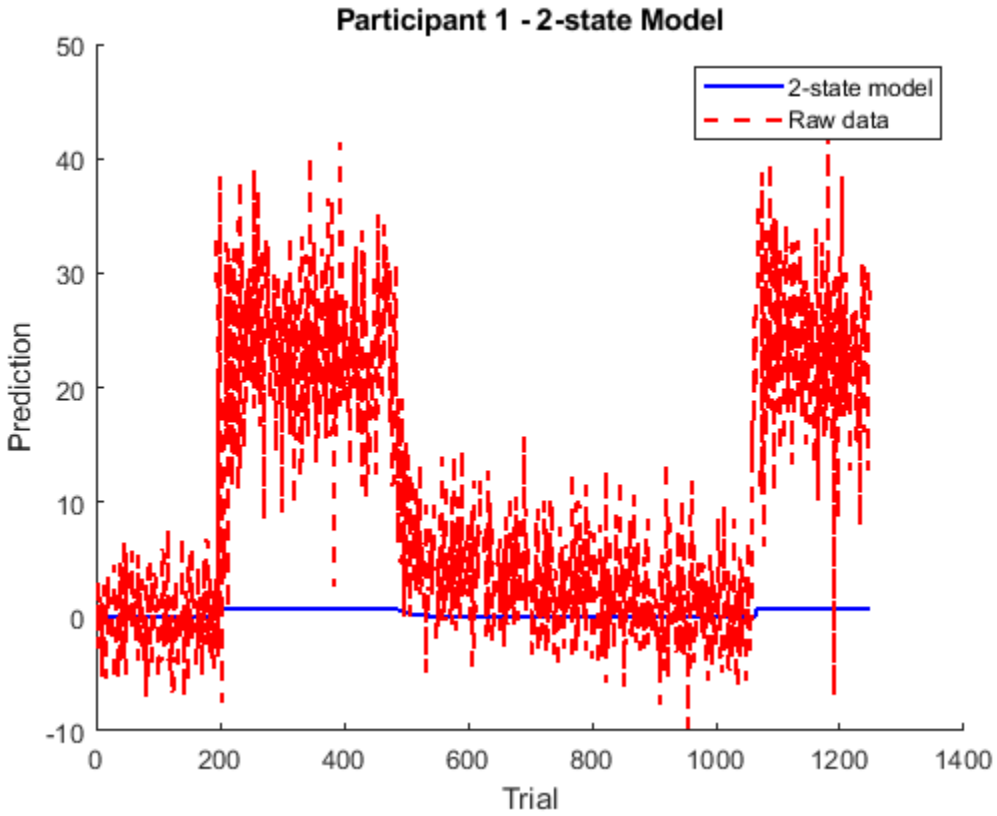
5. **Model comparison: Now you have a working 1-state and 2-state model. First return the 1-state and 2-state model to original values:**

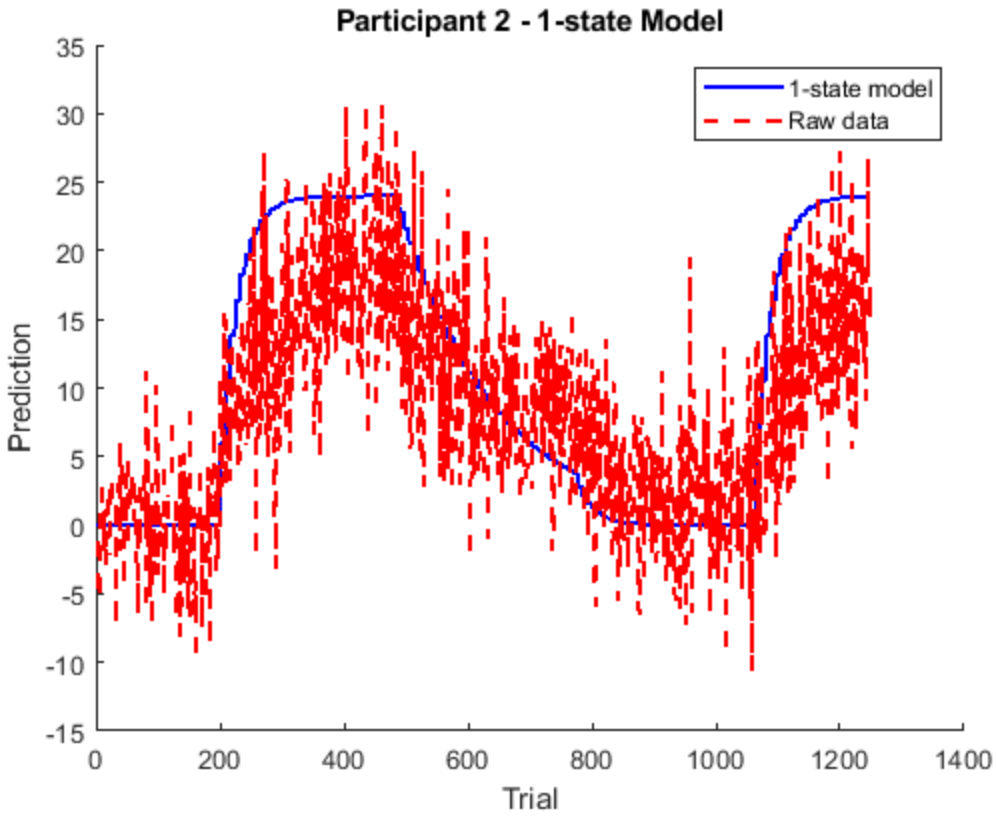
   1-state = M.A=0.95, M.B=0.2.

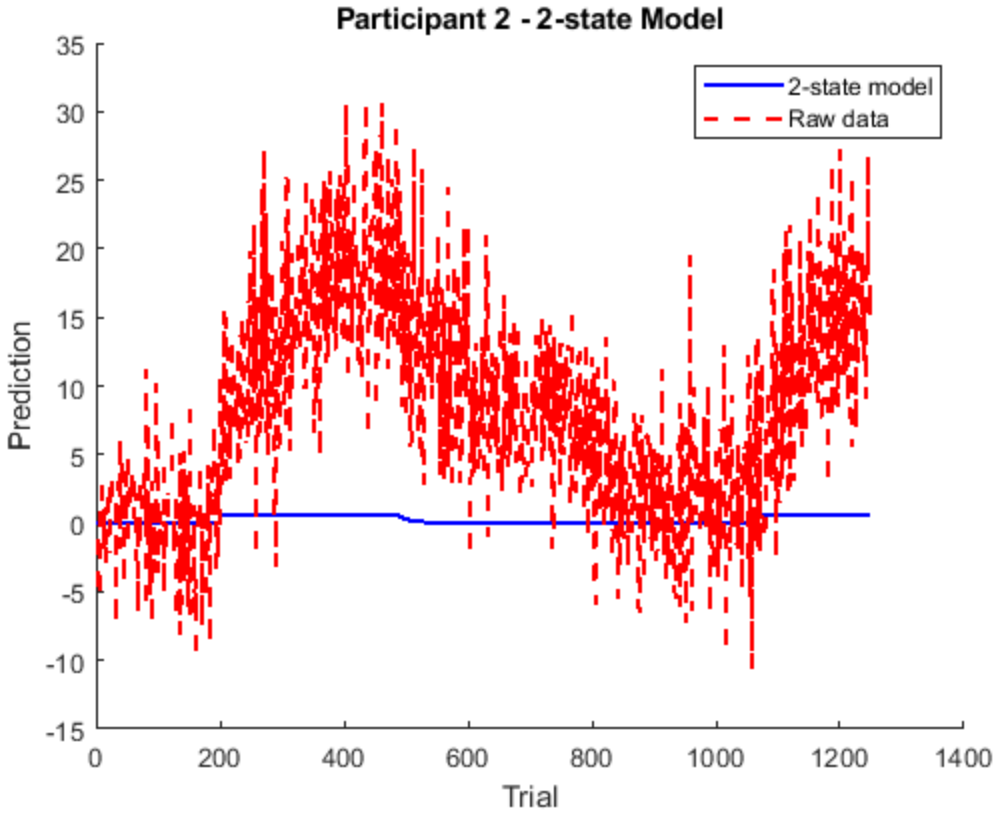
   2-state = M.A = [0.7 0.99] M.B = [0.15 0.02];

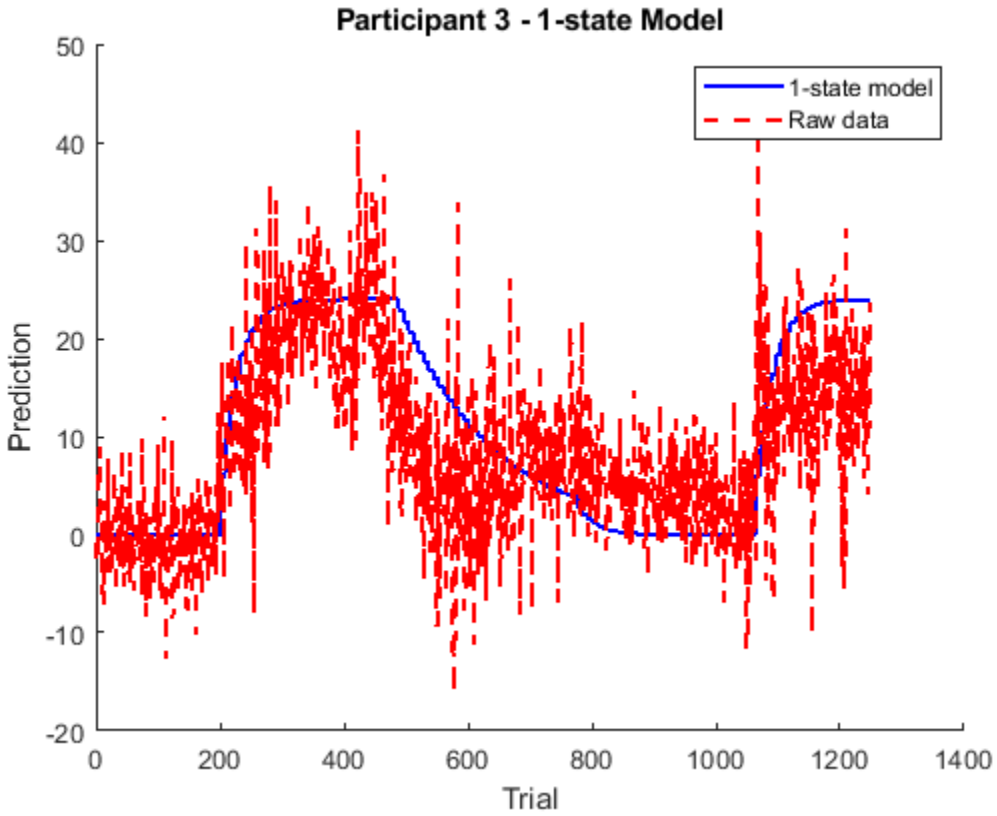
Run the models for each subject. Run each model for each participant. For each participant, create a figure which compares the raw data (y) with the predicted model (yp) from the 1-state and 2-state models. Submit the 5 figures.


Participant 1 - 1-state Model
1-state model
Raw data
Prediction
Trial
50
40
30
20
10
0
-10
0
200
400
600
800
1000
1200
1400


Participant 1 - 2-state Model
2-state model
Raw data
Prediction
Trial
50
40
30
20
10
0
-10
0
200
400
600
800
1000
1200
1400


Participant 2 - 1-state Model
1-state model
Raw data
Prediction
Trial
35
30
25
20
15
10
5
0
-5
-10
-15
0
200
400
600
800
1000
1200
1400


Participant 2 - 2-state Model
2-state model
Raw data
Prediction
Trial
35
30
25
20
15
10
5
0
-5
-10
-15
0
200
400
600
800
1000
1200
1400


Participant 3 - 1-state Model
1-state model
Raw data
Prediction
Trial
50
40
30
20
10
0
-10
-20
0
200
400
600
800
1000
1200
1400

6. **Model fit: Using matlab, could you perform some form of model comparison between these two models using the original values? Submit the details of your attempt at comparing the fit of the 2 models across participants (ie) function used, results.**

   To compare the fit of the 1-state and 2-state models across participants, we use the root mean square error (RMSE). These measures quantify the difference between the predicted values and the actual values (raw data).

   1-state model RMSE:

   7.0577 7.0783 7.8151 6.9109 10.2075

   2-state model RMSE:

   15.2667 11.1682 12.3790 14.8180 7.7837

   The results of the model comparison between the 1-state and 2-state models indicate that the 1-state model provides a better fit to the data for the given participants. The RMSE values, which represent the average error between the model predictions and the actual data values, are consistently lower for the 1-state model compared to the 2-state model across all participants. This suggests that the 1-state model is able to capture the underlying patterns in the data more accurately. The RMSE values for the 1-state model range from approximately 6.911 to 10.208 units, while for the 2-state model, they range from around 7.784 to 15.267 units. These findings suggest that the 1-state model is more effective in explaining the observed visuomotor adaptation compared to the 2-state model.

7. **Adding a third state: Finally, can you add a third state to the 2-state model? Does this improve the fits across participants (using the same logic from question 5)? Submit the code which incorporates the third state and any model comparison data.**

   Here's the code that incorporates the third state:

```
case 'fit_all_ts'
    load('T.mat')
    D = T;
    T = [];
    M.A = [0.7 0.99 0.8];  % Update A to include a third state
    M.B = [0.15 0.02 0.03];  % Update B to include a third state
    M.z0 = [0 0 0 0 0 0 0 0];
    M.LB = [0 0 -1 -1 -1]';  % Update LB to include a third state
    M.UB = [1 1 1 1 1]';  % Update UB to include a third state
    for s = 1
        this = find(D.SN == s);
        D = getrow(D, this);
        N = length(D.u);
        zf = M.z0;
        zs = M.z0;
        zt = M.z0;  % Initialize the third state
        for n = 1:N-1
            y(n,1) = -zf(n,D.targetnum(n)) - zs(n,D.targetnum(n)) -
zt(n,D.targetnum(n));  % Update y to include the third state
            zf(n+1,:) = zf(n,:);
            zs(n+1,:) = zs(n,:);
            zt(n+1,:) = zt(n,:);
            if isnan(D.u(n))
                zf(n+1,D.targetnum(n)) = M.A(1) *
zf(n,D.targetnum(n));
                zs(n+1,D.targetnum(n)) = M.A(2) *
zs(n,D.targetnum(n));
                zt(n+1,D.targetnum(n)) = M.A(3) *
zt(n,D.targetnum(n));  % Update the update equation for the third
state
            elseif D.feedback(n) == 0
                zf(n+1,D.targetnum(n)) = M.A(1) *
zf(n,D.targetnum(n));
                zs(n+1,D.targetnum(n)) = M.A(2) *
zs(n,D.targetnum(n));
                zt(n+1,D.targetnum(n)) = M.A(3) *
zt(n,D.targetnum(n));  % Update the update equation for the third
state
            else
```

```
                zf(n+1,D.targetnum(n)) = M.A(1) *
zf(n,D.targetnum(n)) + M.B(1) * (D.u(n) - zf(n,D.targetnum(n)) -
zs(n,D.targetnum(n)) - zt(n,D.targetnum(n)));  % Update the update
equation for the third state
                zs(n+1,D.targetnum(n)) = M.A(2) *
zs(n,D.targetnum(n)) + M.B(2) * (D.u(n) - zf(n,D.targetnum(n)) -
zs(n,D.targetnum(n)) - zt(n,D.targetnum(n)));  % Update the update
equation for the third state
                zt(n+1,D.targetnum(n)) = M.A(3) *
zt(n,D.targetnum(n)) + M.B(3) * (D.u(n) - zf(n,D.targetnum(n)) -
zs(n,D.targetnum(n)) - zt(n,D.targetnum(n)));  % Update the update
equation for the third state
            end
        end
        y(N,1) = -zf(N,D.targetnum(N)) - zs(N,D.targetnum(N)) -
zt(N,D.targetnum(N));  % Update y to include the third state
        z = [mean(zf,2) mean(zs,2) mean(zt,2)];  % Include the third
state in the z vector
        if M.A(1) > M.A(2)
            M.A = fliplr(M.A);
            M.B = fliplr(M.B);
            z = fliplr(z);
        end
        F.A = M.A;
        F.B = M.B;
        F.z0 = M.z0;
        F.yp = y;
        F.state_fast = z(:,1);
        F.state_slow = z(:,2);
        F.state_third = z(:,3);  % Include the third state in the
output structure
        F.D = D;
        F.y = D.delta_y;
    end
    varargout = {F};
```