

**Cerebellum & Motor Control**

**Implantation of State Space Models**

**Bradley Morgan 1731302**

|  |
| --- |
| **Implementation: Single State Space Model** |
| **Relevant Code** |
| State 1: z(n+1,D.targetnum(n))=M.A\*z(n,D.targetnum(n)) + (M.B \* (D.u(n) - z(n,D.targetnum(n |
| **Parameters** |
| State 1 A = 0.95  State 1 B = 0.2 |
| **Visualisation** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q1 figs\ss_model.jpg** |

|  |
| --- |
| **Implementation: Double State Space Model** |
| **Relevant Code** |
| State 1: 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)));  State 2: 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))); |
| **Parameters** |
| Fast A = 0.7  Slow A = 0.84  Fast B = 0.4  Slow B = 0.3 |
| **Visualisation** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q2 figs\ts_model.jpg** |

|  |  |
| --- | --- |
| **Double State Space Model: Varying the Parameters** | |
| **Parameters** | **Parameters** |
| Fast A = 0.7  **Slow A = (0.99 – 0.5)**  Fast B = 0.15  Slow B = 0.02 | **Fast A = (0.7 – 0.5)**  Slow A = 0.99  Fast B = 0.15  Slow B = 0.02 |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q3 figs\1.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q3 figs\2.jpg** |
| **Parameters** | **Parameters** |
| Fast A = 0.7  Slow A = 0.99  **Fast B = (0.15 + 0.5)**  Slow B = 0.02 | Fast A = 0.7  Slow A = 0.99  Fast B = 0.15  **Slow B = (0.02 + 0.5)** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q3 figs\3.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q3 figs\4.jpg** |

|  |  |
| --- | --- |
| **Model Fit Comparison: Single vs Double State Space Model** | |
| **Subject1** | **Subject2** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q4 figs\sub1.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q4 figs\sub2.jpg** |
| **Subject3** | **Subject4** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q4 figs\sub3.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q4 figs\sub4.jpg** |
| **Subject5** | **Error Between Models and The Raw Data: All Subjects** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q4 figs\sub5.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q4 figs\Error.jpg** |

**Calculating Model Error & Parameter Selection: Parameter Space Search Program**

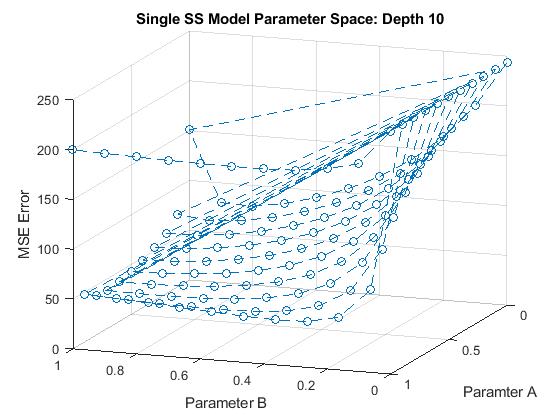
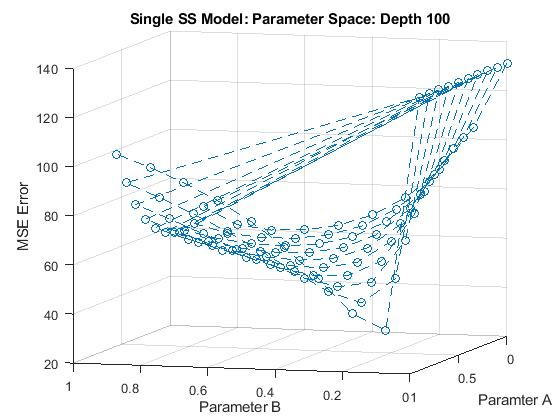
The Single, double and Triple State Space Model performance was assessed by calculating the mean squared error of the Model prediction vector against that of the Raw data vector. The cost function ignores NaN data points. The formula is as follow, where is the observed data and the model prediction:

Parameters were selected by searching parameter space at varying depths, the procedure was as follows:

1. Initiate the Parameter Space Searcher Program. This program will iterate through all possible configurations of A-B or FastA-FastB-Slow-A-Slow-B parameters.
2. The program will pass current parameters to either the single or double state space model program, and in turn retrieve a model prediction.
3. The Program then calculates and records each configuration mean squared error at firstly depth 10 for each subject dataset. Depth refers to which decimal place the parameters will be tuned. For example, given A = 0.100, B= 0.201, a depth of 10 will cause the program to only vary the digits one decimal place to the right for each parameter.
4. The program the finds the parameter combination that produced the minimum amount of error for the current subject and depth value. These parameter settings are stored.
5. The program then increases the depth by a factor of 10. The program then repeats steps 1-4. The program Uses the optimal parameters found in the previous depth space as the bounds for the new search. For examples optimal A = 0.20, B = 0.90 depth=10, now at depth = 100 the program will vary the digits that are 2 decimal places to the right.
6. Repeat step 5 until satisfied with the level of precision.
7. The program then takes parameters that produced the least error across all depths, per subject. The program the averages the parameters values across all subjects to produce a set of parameters values that aim to generalise reasonably across all given datasets.

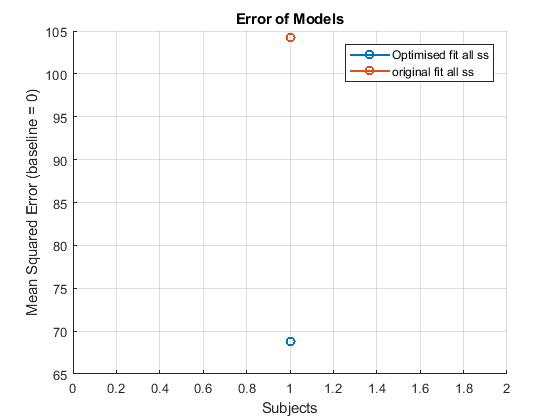
The Code Can be Found in the appendix below:

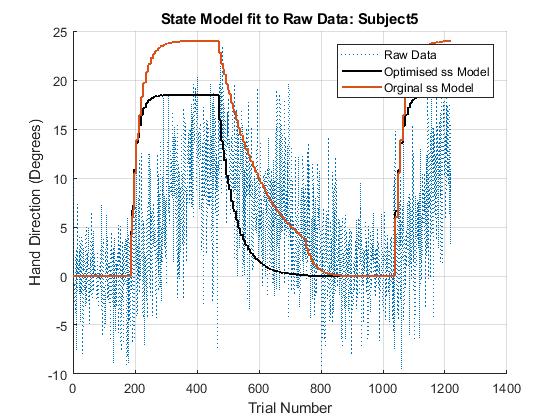
**Plots** **: Parameter Space Search Program & Single State Model Optimisation**



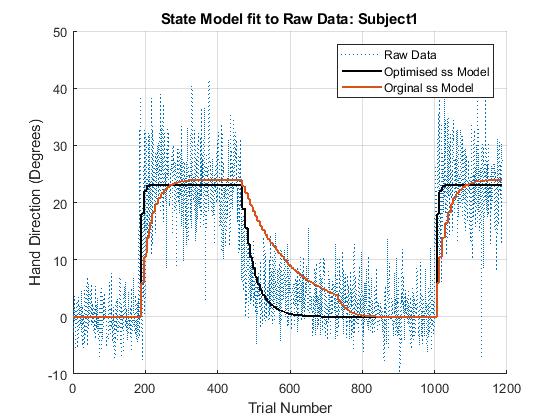
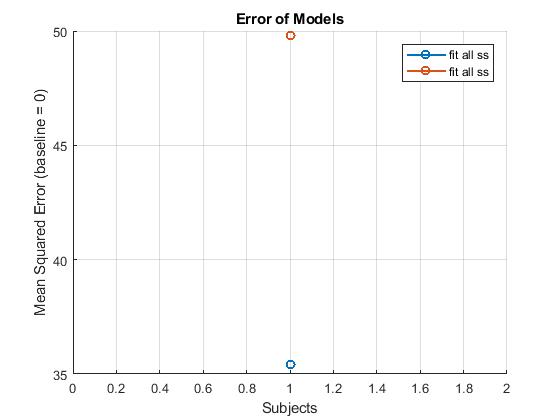
*The Above example plots are from using Subject 5 dataset and model Predictions*

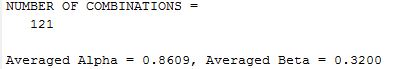
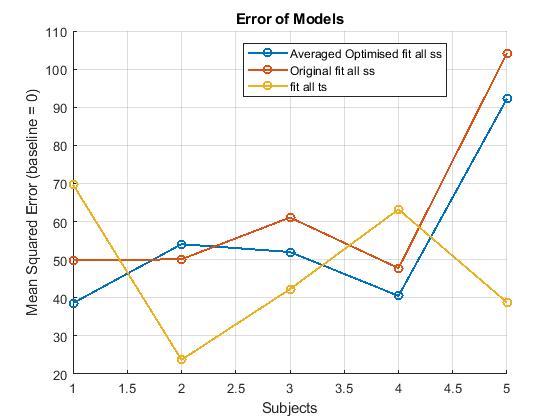
First, I searched for the lowest parameter data point on the error gradient at depth 10. I Found that Searching Parameter Space at depth 100 yielded then steepest error gradient, Thus the lowest error with respect to the parameter values. The gradient at depth 1000 became relatively flat, and therefore didn’t really effect the overall model error too much, when selecting parameter values.

*All Optimised Single State Models were compared against a single SS model with the original parameter values A = 0.95, B = 0.2.* ***Optimised Subject 5 : Alpha = 0.8300, Beta = 0.1000 Subject1 : Alpha = 0.8200, Beta = 0.6000***



As shown Above the Parameter Space Search Program was able to improve model performance on difficult datasets to fit towards. Also as shown below the program was able to tighten accuracy of the model fit on datasets it already performed reasonably well at, using the original parameter values.

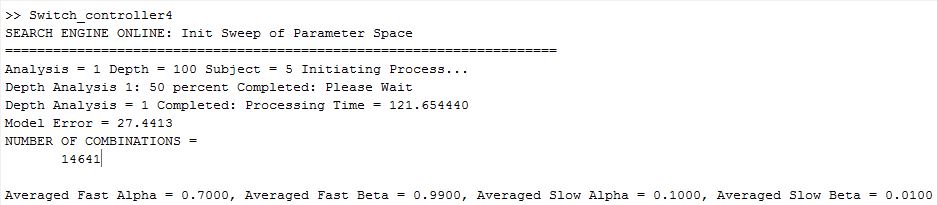
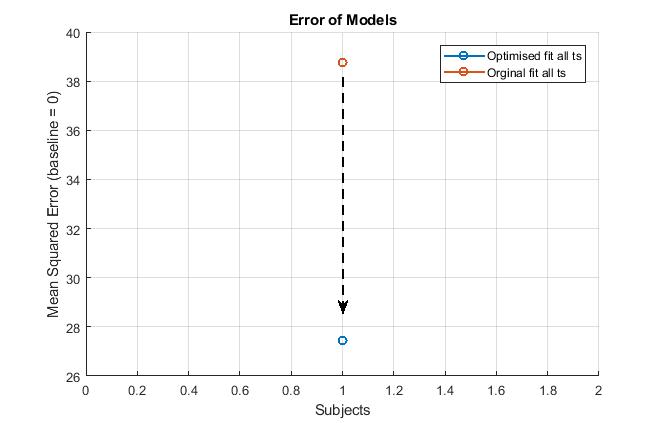


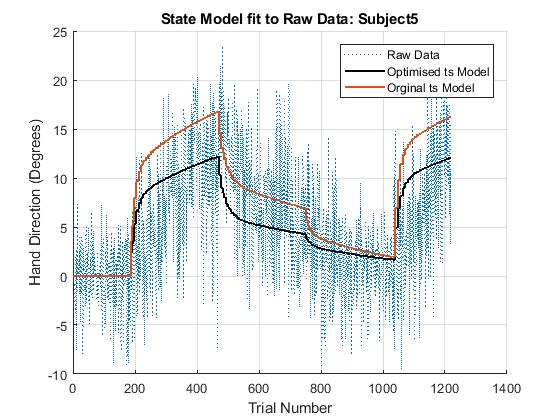
**Using Averaged Parameter values**

As Shown by the Error Figure to the right, using an average of optimised values across subjects did overall improve the performance of the Single SS model from the original parameter set up. Nonetheless as Shown above performance of the Single SS model is much better when optimised subject specific parameter values are used, as compared to averaged optimised values. However, the unoptimized double SS model, still outperforms the optimised single SS model on subjects 3, 2 and 5 datasets. This suggests that much more accurate fitting will be gained by using and optimising a double SS model rather than a single state model.

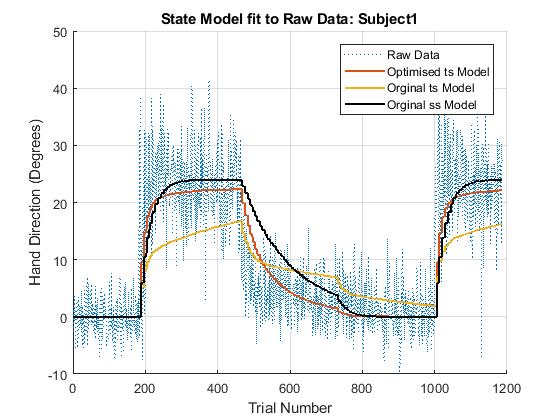
**Plots: Parameter Space Searcher Program & Double State Model Optimisation**

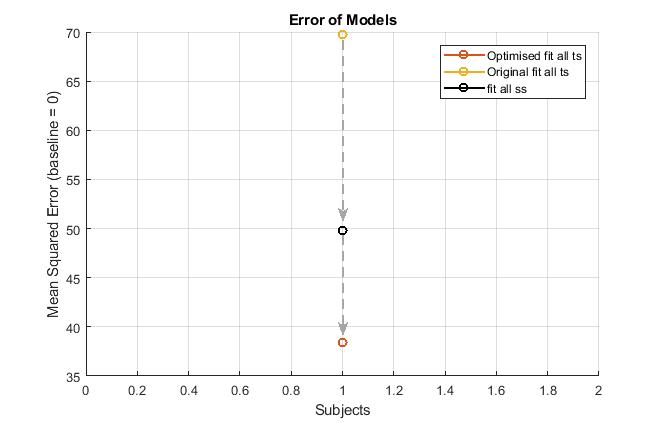
Optimised Fitting Subject5 Dataset:





Optimised Fitting Subject1 Dataset:





|  |  |
| --- | --- |
| **Model Fit Comparison: Single vs Double vs Triple State Space Model** | |
| **Subject1** | **Subject2** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q6\subject1.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q6\Subject2.jpg** |
| **Subject3** | **Subject4** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q6\Subject3.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q6\Subject4.jpg** |
| **Subject5** | **Error Between Models and The Raw Data: All Subjects** |
| **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q6\Subject5.jpg** | **C:\Users\Novus\Desktop\Mind_Brain_Model\Workshop 5\Report\Q6\Subject6.jpg** |

%Single State model

ss\_retention\_rate = 0.95;

ss\_forget\_rate = 0.2 ;

% Triple Single state Model

ts3\_retention\_rate = [0.7, 0.99, 0.3];

ts3\_forget\_rate = [0.15, 0.02, 0.2];

% Double Single state Model

ts\_retention\_rate = [0.7, 0.99];

ts\_forget\_rate = [0.15, 0.02];

**Appendix: Triple State Space Model Code**:

case 'fit all 3ts'

% Orginal Values

% M.A=[0.7 0.99];

% M.B=[0.15 0.02];

load ('T.mat')

D=T;

T=[];

M.A=retain\_forget(1, :);

M.B=retain\_forget(2, :);

M.z0=[0 0 0 0 0 0 0 0];

M.LB=[0 0 -1 -1 ]';

M.UB=[1 1 1 1 ]';

for s=subject

this=find(D.SN==s);

D=getrow(D,this);

N=length(D.u);

zf=M.z0;

zs=M.z0;

zq =M.z0;

for n=1:N-1

y(n,1)=-zf(n,D.targetnum(n))-zs(n,D.targetnum(n))-zq(n,D.targetnum(n));

zf(n+1,:)=zf(n,:);

zs(n+1,:)=zs(n,:);

zq(n+1,:)=zq(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));

zq(n+1,D.targetnum(n))=M.A(3)\*zq(n,D.targetnum(n));

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));

zq(n+1,D.targetnum(n))=M.A(3)\*zq(n,D.targetnum(n));

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))-zq(n,D.targetnum(n))); %**

**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))-zq(n,D.targetnum(n)));**

**zq(n+1, D.targetnum(n))=M.A(3)\*zq(n, D.targetnum(n))+M.B(3)\*(D.u(n)-zf(n,D.targetnum(n))-zs(n,D.targetnum(n))-zq(n,D.targetnum(n)));**

end

end;

y(N,1)=-zf(N,D.targetnum(N))-zs(N,D.targetnum(N))-zq(N,D.targetnum(N));

z = [mean(zf,3) mean(zs,3) mean(zq,3)];

if (M.A(1)>M.A(2) || M.A(1)>M.A(3))

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\_extra=z(:,3);

F.D=D;

F.y=D.delta\_y;

end

varargout={F};

**Appendix: Mean Squared Error**

function [varargout] = MSE(observed, prediction)

%check the sizes are equal

[xr, xc] = size(observed);

[yr, yc] = size(prediction);

if ~isequal(xr, yr) || ~isequal(xc, yc)

error('ERROR: X and Y Inputs Must Be the Same Size')

end

error\_store = zeros(xr \* xc, 1);

nan\_count = 0;

for j = 1:yc

for k = 1:yr

pbit = prediction(k, j);

obit = observed(k, j);

if isnan(pbit) || isnan(obit)

nan\_count = nan\_count + 1;

error\_store(k, j) = NaN;

else

error\_store(k, j) = (obit - pbit)^2;

end

end

end

error\_store = reshape(error\_store, (yr \* yc), 1);

error\_store(isnan(error\_store)) = [];

if nargout == 1

varargout{1} = mean(error\_store);

elseif nargout == 2

varargout{1} = mean(error\_store);

varargout{2} = nan\_count;

end

end

**Appendix B: Parameter Space Search Program**

Single State Space Optimiser: *(The Double State Optimiser has too extra Configuration loops one for extra dimensionality of Slow A and Slow B, the rest is exactly the same.*

switches = [0 1 2 3 4 5 6 7 8 9 10 ];

store = [];

depth = [10 100 1000];

subjects = [ 1 2 3 4 5];

% Change 1

terminate = 0;

trigger = 0;

limiter = 0;

limit = 10;

ErrorBox = cell(length(depth),length(subjects));

fixed\_digits\_store = zeros(length(depth), 2);

disp('SEARCH ENGINE ONLINE: Init Sweep of Parameter Space')

**for S = 1:length(subjects)**

fixed\_digits = [NaN NaN];

**for D = 1:length(depth)**

disp('=====================================================================')

txt = 'Analysis = %1.0f Depth = %1.0f Subject = %1.0f Initiating Process...\n';

fprintf(txt, D, depth(D), subjects(S))

c1 = 1;

tic;

if isequal(limiter, 1)

limit = 9;

end

**CONFIGURATION LOOPS**

**for A = switches:limit**

**for B = switches:limit**

if ~isnan(fixed\_digits)

a = fixed\_digits(1) +(A/depth(D));

b = fixed\_digits(2) +(B/depth(D));%Change

else

a = A/depth(D);

b = B/depth(D);

%a = [A, B]/depth(D);

end

store(c1, :) = a;%Change

c1 = c1 +1;

%disp([i,j]) %Change

% try Combination

model\_output = state\_space\_lecture\_3('fit all ss' ,[a;b], subjects(S));

[error, detect\_nan] = MSE(model\_output.D.delta\_y, model\_output.yp);

ErrorBox{D, S}(c1-1, 1) = error;

ErrorBox{D, S}(c1-1, 2) = a;

ErrorBox{D, S}(c1-1, 3) = b;

if isequal(A,10) && isequal(trigger, 0)

trigger = 1;

formatspec = 'Depth Analysis %1.0f: 50 percent Completed: Please Wait \n';

fprintf(formatspec, D)

end

end

end

[M,I] = min(ErrorBox{D, S}(:, 1));

fixed\_digits(1) = ErrorBox{D, S}(I, 2);

fixed\_digits(2) = ErrorBox{D, S}(I, 3);

fixed\_digits\_store(D, 1:2) = ErrorBox{D, S}(I, 2:3);

fixed\_digits\_store(D, 3) = M;

fixed\_digits\_store(D, 4) = I;

limiter = 1;

time = toc;

fprintf('Depth Analysis = %1.0f Completed: Processing Time = %1.6f \n', D, time)

fprintf('Model Error = %1.4f \n', M)

end

[M,I] = min(fixed\_digits\_store(:, 3));

Paramters(S, :) = fixed\_digits\_store(I, 1:2);

end

average\_paramtersA = mean(Paramters(:, 1));

average\_paramtersB = mean(Paramters(:, 2));

disp('NUMBER OF COMBINATIONS = ')

disp(length(store))

fprintf('Averaged Alpha = %1.4f, Averaged Beta = %1.4f \n',average\_paramtersA, average\_paramtersB);