Problem 6;

Here is the code that I used to attempt this problem. This is for the DAM case. For some reason, I am still getting very high error probabilities resulting from large amounts of errors after a single error. The error probability seems to be slightly less than 0.5 which is not much better than just random numbers, so I concluded that this is probably incorrect.

num\_bits = 100000;

var = 1;

Eb\_dB = -5:5;

Eb = var \* 10.^(Eb\_dB / 10);

A = sqrt(Eb);

%A = [1 2 3 4 5 6 7 8 9 10 11];

state = 0;

%incoming bits.

incoming = floor(rand(1, num\_bits) + 0.5);

%error correction code.

conv = zeros(1, 2\*num\_bits);

for i = 1:num\_bits

if state == 0

if incoming(i) == 0

conv(2\*i-1) = 0;

conv(2\*i) = 0; %technically unnecessary.

state = 0; %technically unnecessary.

else

conv(2\*i-1) = 1;

conv(2\*i) = 1;

state = 1;

end

elseif state == 1

if incoming(i) == 0

conv(2\*i-1) = 0;

conv(2\*i) = 1;

state = 2;

else

conv(2\*i-1) = 1;

conv(2\*i) = 0;

state = 3;

end

elseif state == 2

if incoming(i) == 0

conv(2\*i-1) = 1;

conv(2\*i) = 0;

state = 0;

else

conv(2\*i-1) = 0;

conv(2\*i) = 1;

state = 1;

end

else

if incoming(i) == 0

conv(2\*i-1) = 1;

conv(2\*i) = 1;

state = 2;

else

conv(2\*i-1) = 0;

conv(2\*i) = 0;

state = 3; %technically unnecessary.

end

end

end

%A(1) = 10;

%bit to symbol

%here we use 0 to -A and 1 to A.

prob\_error = zeros(1, 11);

for j = 1:1

symbols = A(j)\*(2\*conv - 1);

noise = randn(1, 2\*num\_bits);

receive = symbols + noise;

min\_dist = zeros(4, num\_bits + 1);

prev\_path = zeros(4, num\_bits + 1);

%maintenance to make sure that the minpath does not

%start with a state other than zero.

min\_dist(:,1) = [0 1000 1000 1000];

prev\_path(:,1) = [-1 -1 -1 -1];

for i = 1:num\_bits

y0 = receive(2\*i-1);

y1 = receive(2\*i);

path00 = (y0 + A(j))^2 + (y1 + A(j))^2;

path01 = (y0 - A(j))^2 + (y1 - A(j))^2;

path13 = (y0 - A(j))^2 + (y1 + A(j))^2;

path12 = (y0 + A(j))^2 + (y1 - A(j))^2;

path21 = path12;

path20 = path13;

path32 = path01;

path33 = path00;

if min\_dist(1,i)+path00 < min\_dist(3,i)+path20

min\_dist(1,i+1) = min\_dist(1,i)+path00;

prev\_path(1,i+1) = 0;

else

min\_dist(1,i+1) = min\_dist(3,i)+path20;

prev\_path(1,i+1) = 2;

end

if min\_dist(3,i)+path21 < min\_dist(1,i)+path01

min\_dist(2,i+1) = min\_dist(3,i)+path21;

prev\_path(2,i+1) = 2;

else

min\_dist(2,i+1) = min\_dist(1,i)+path01;

prev\_path(2,i+1) = 0;

end

if min\_dist(4,i)+path32 < min\_dist(2,i)+path12

min\_dist(3,i+1) = min\_dist(4,i)+path32;

prev\_path(3,i+1) = 3;

else

min\_dist(3,i+1) = min\_dist(2,i)+path12;

prev\_path(3,i+1) = 1;

end

if min\_dist(2,i)+path13 < min\_dist(4,i)+path33

min\_dist(4,i+1) = min\_dist(2,i)+path13;

prev\_path(4,i+1) = 1;

else

min\_dist(4,i+1) = min\_dist(4,i)+path33;

prev\_path(4,i+1) = 3;

end

end

[m,index] = min(min\_dist(:,num\_bits+1));

path = zeros(1, num\_bits);

path(num\_bits) = index - 1;

for i=0:num\_bits-2

path(num\_bits-1-i) = prev\_path(index, num\_bits+1-i);

index = prev\_path(index, num\_bits+1-i) + 1;

end

decode = zeros(1, num\_bits);

state = 0;

for i = 1:num\_bits

if state == 0 || state == 2

if path(i) == 0

decode(i) = 0;

else

decode(i) = 1;

end

elseif state == 1 || state == 3

if path(i) == 2

decode(i) = 0;

else

decode(i) = 1;

end

end

state = path(i);

end

prob\_error(j) = mean(incoming ~= decode);

end