We are tasked with implementing the DPCM system. First, we need to determine the values we will use, such as the order of the predictor (how many samples it will have for prediction) and the levels of the quantizer. For the initial implementation, I use **p = 3** and **N = 10**, along with the dynamic range provided by the exercise (-3.5, 3.5).

```
function main()

close all;

% posa proigoumena deigmata tha exoume
p = 10;
% ta bit kvadisti
N = 3;
% dinamiki perioxi tou kvadisti
min_value = -3.5;
max_value = 3.5;
load('source.mat')
```

Next, before encoding, it is necessary to solve the Yule-Walker equation to calculate the prediction sample.

```
R = zeros(p,p);
r = zeros(p,1);
```

We initialize the two variables **r** and **R** (which are actually matrices) and proceed with their calculation:

```
for i=1:p
    sum_r = 0;
    for n=p+1:sinolikos_arithmos_deigmaton
        sum_r = sum_r + x(n)*x(n-i);
    end
    r(i) = sum_r * 1/(sinolikos_arithmos_deigmaton-p);
    for j=1:p
        sum_R = 0;
        for n=p+1:sinolikos_arithmos_deigmaton
            sum_R = sum_R + x(n-j)*x(n-i);
        end
        R(i,j) = sum_R * 1/(sinolikos_arithmos_deigmaton-p);
```

Finally, we solve the equation:

```
a = R\r;
```

We then calculate the quantized values of the coefficients with the appropriate quantizer values:

```
for i=1:p
    a(i) = my_quantizer(a(i), 8, -2, 2);
```

Next, we are now able to perform the encoding. The prediction is calculated with the **a** values we found and the values of the previous quantized samples. The prediction error is the current sample minus the prediction. Finally, we quantize the prediction error:

```
for i=1:sinolikos_arithmos_deigmaton
    y(i) = x(i) - y_meto_simvolo_apano_tonismeno;
    y_meto_simvolo_apano(i) =
my_quantizer(y(i),N,min_value,max_value);


 y_meto_simvolo_apano_tonismeno = y_meto_simvolo_apano(i) +
y_meto_simvolo_apano_tonismeno;


 proigoumena_deigmata = [y_meto_simvolo_apano_tonismeno;
proigoumena_deigmata(1:p-1)];


 y_meto_simvolo_apano_tonismeno = a'*proigoumena_deigmata;
end
```

The loop performs the above process and also adjusts the prediction for the next sample based on the quantization error of the current sample, thus improving the predictor.

Immediately after, we implement the decoder. In simple terms, we will perform the reverse process of encoding, meaning that from the quantized prediction errors, we will reconstruct the original signal.

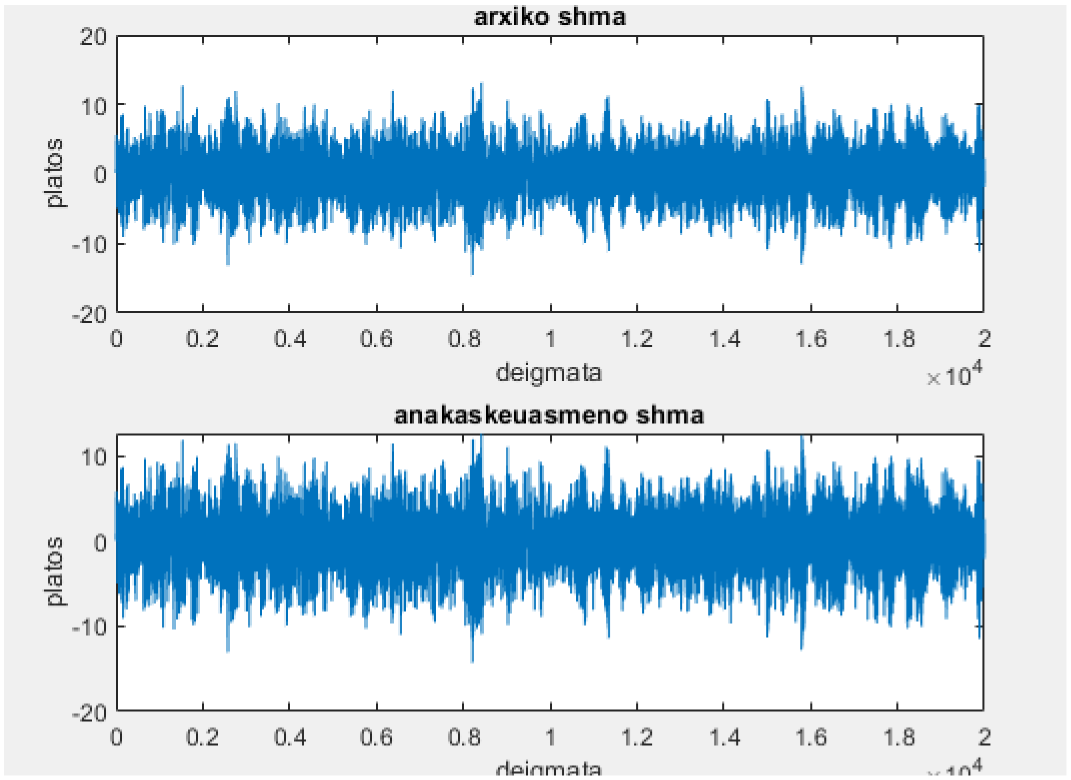
```
% apokodikopoihths
proigoumena_deigmata = zeros(p,1);
y_meto_simvolo_apano_tonismeno = 0;
anakataskeuasmeno_sima = zeros(sinolikos_arithmos_deigmaton,1);
for i=1:sinolikos_arithmos_deigmaton
    anakataskeuasmeno_sima(i) = y_meto_simvolo_apano(i) +
y_meto_simvolo_apano_tonismeno;
    proigoumena_deigmata = [anakataskeuasmeno_sima(i);
proigoumena_deigmata(1:p-1)];
    y_meto_simvolo_apano_tonismeno = a'*proigoumena_deigmata;
end
```

The only thing left to do is to implement the quantizer. We need to find the step **Δ**, define the quantization centers, and assign each value to the appropriate level.

```
function y_final = my_quantizer(y, N, min_value, max_value)

% periorizoume to deigma mesa sta oria tou kvadisti
if y < min_value
    y = min_value;
```

```
end
if y > max_value
    y = max_value;
end

% to step size tou kvadisth
D = (max_value - min_value) / (2^N - 1);

% kedra
centers = zeros(2^N, 1);
for i = 1:2^N
    centers(i) = min_value + D * (i - 1);
end

% se pio epipedo tha paei to deigma
for i = 1:2^N
    if (y >= centers(i) - D/2) && (y <= centers(i) + D/2)
        y_final = centers(i);
        break;
    end
end

end
```

With the indicative values of **N** and **p**, we observe that the system operates normally.


arxiko shma
platos
deigmata
×10^4
anakaskeuasmeno shma
platos
deigmata