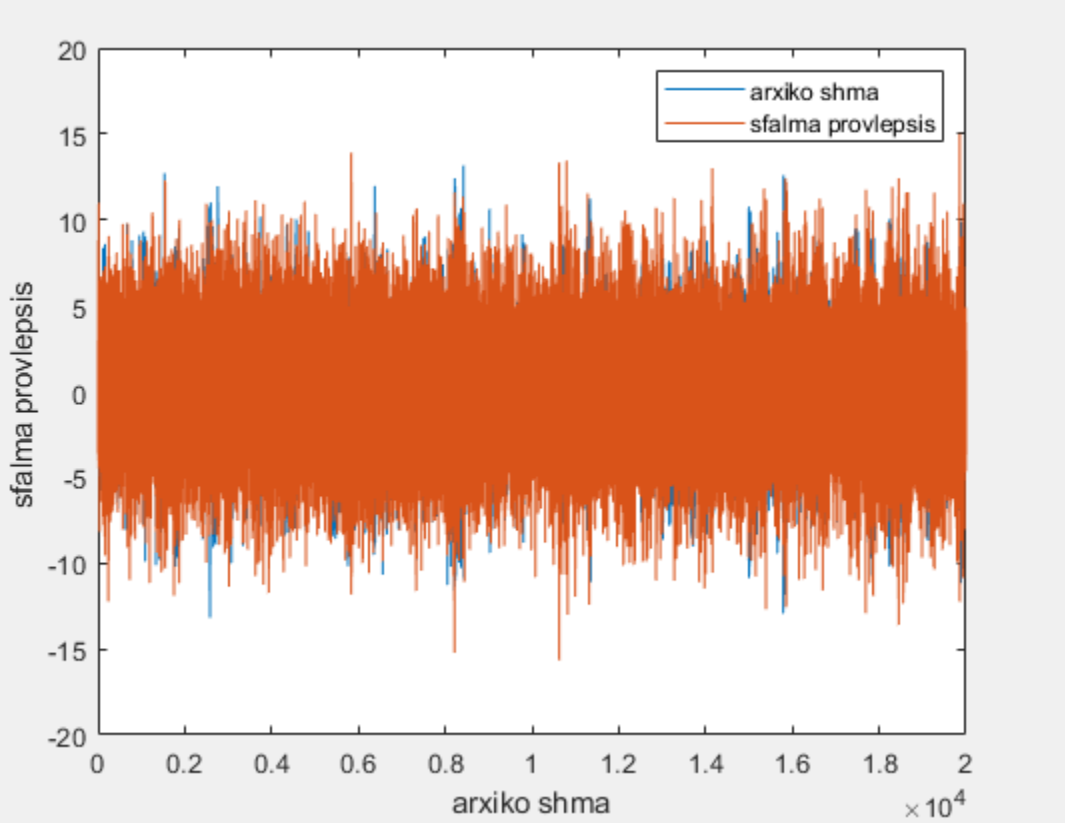
I choose the values **p = 6, 12**. The only thing we need to do in the code is to set the appropriate values for **p** and **N** each time, and add the following plot commands:

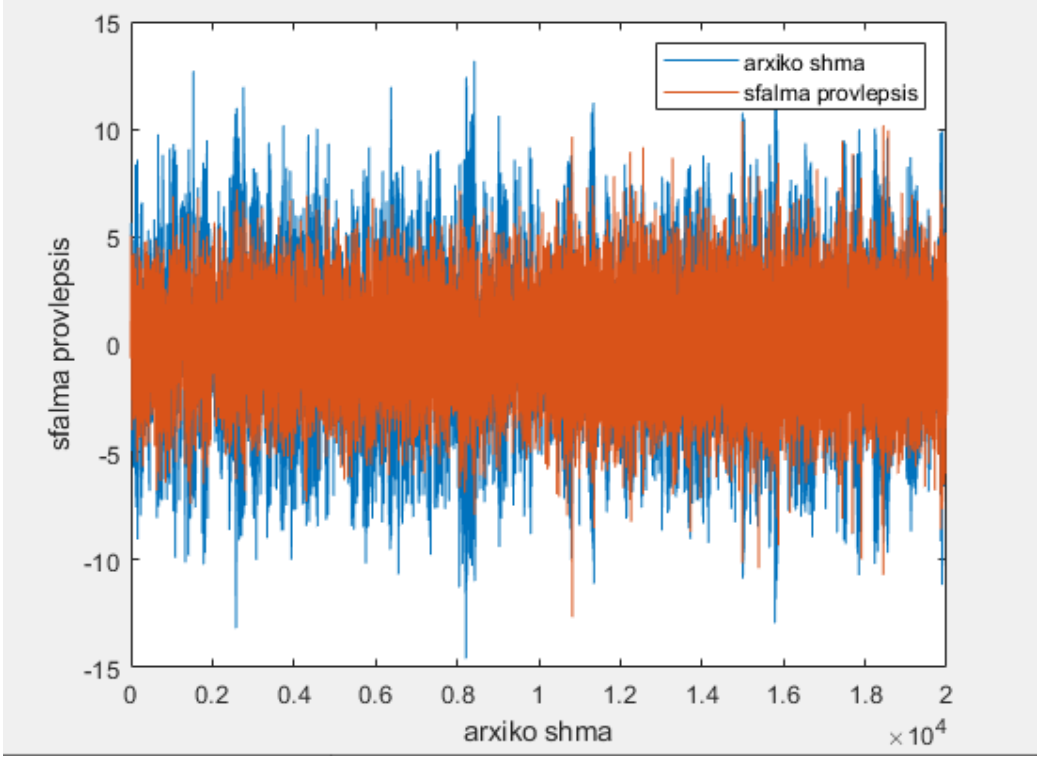
```
plot(t)
     hold on
     plot(y)
     xlabel('arxiko shma')
     ylabel('sfalma provlepsis ')
     legend('arxiko shma', 'sfalma provlepsis')
     hold off
```

P=6 :

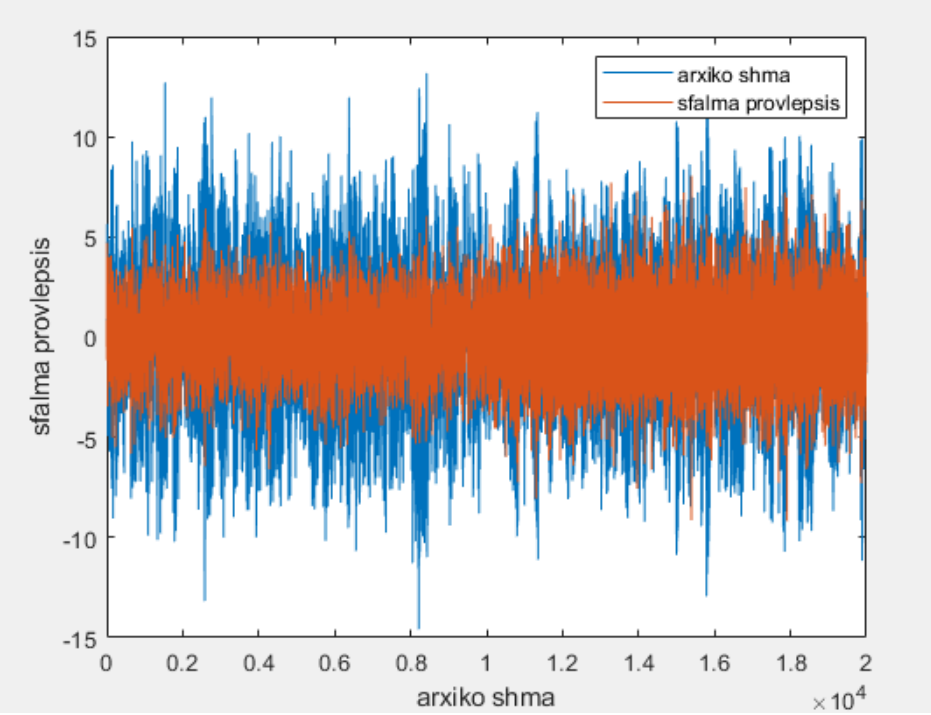
N=1



N=2

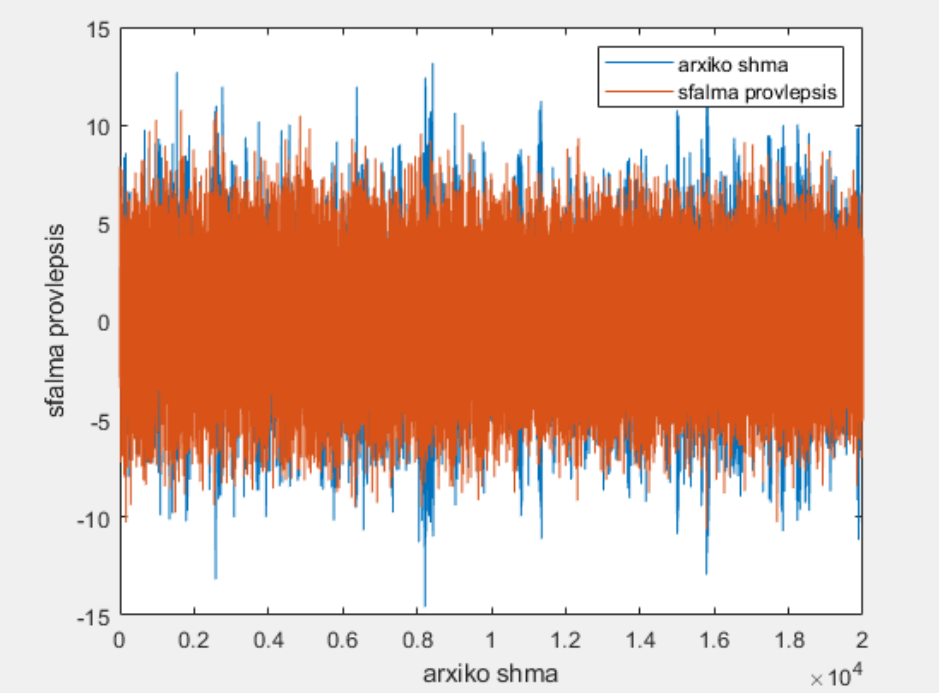

arxiko shma
sfalma provlepsis
sfalma provlepsis
arxiko shma
15
10
5
0
-5
-10
-15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4

N=3

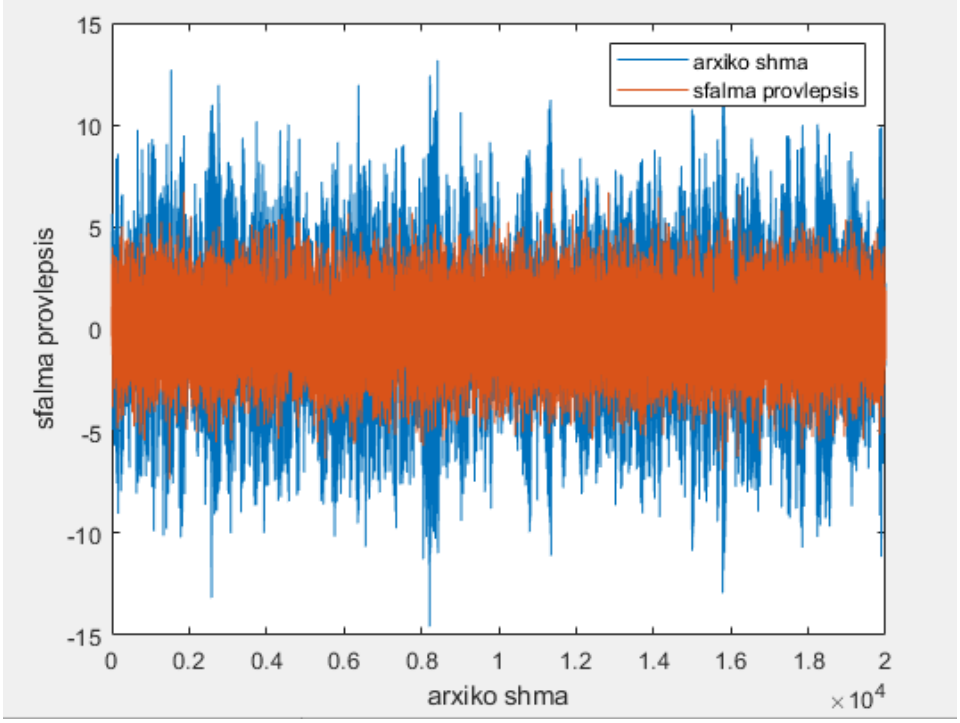

arxiko shma
sfalma provlepsis
sfalma provlepsis
arxiko shma
15
10
5
0
-5
-10
-15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4

P=12 :

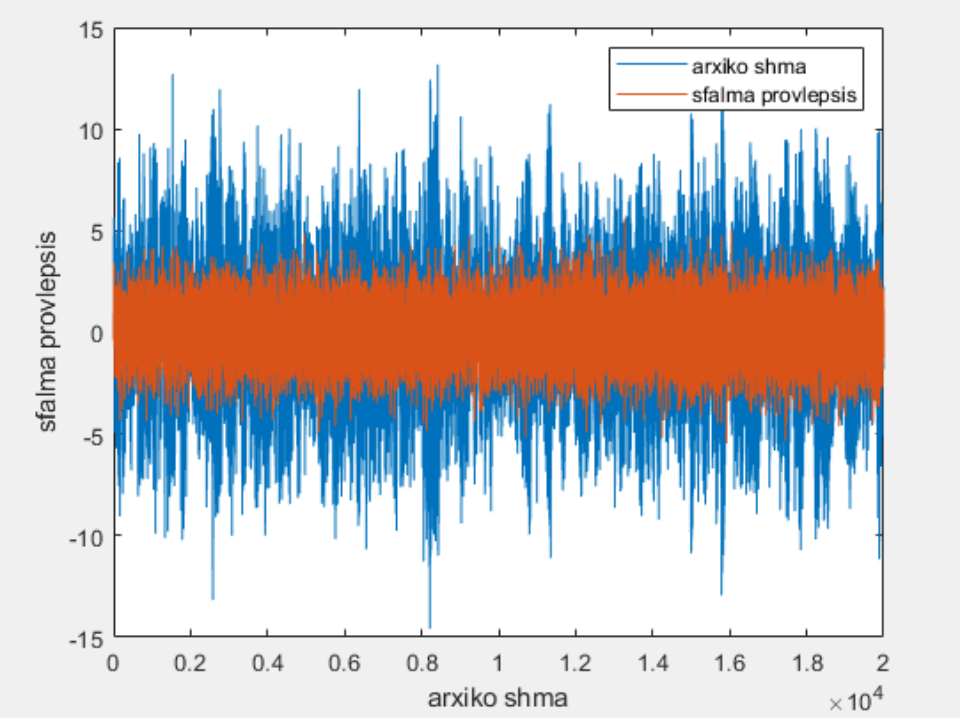
N=1


arxiko shma
sfalma provlepsis
sfalma provlepsis
arxiko shma
15
10
5
0
-5
-10
-15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4

N=2


arxiko shma
sfalma provlepsis
sfalma provlepsis
arxiko shma
15
10
5
0
-5
-10
-15
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
×10^4

N=3



We can easily distinguish that as **N** increases, the prediction error decreases. This result is logical, as with more levels, the step **Δ** of the quantizer decreases, thus reducing the difference between the prediction and the current value. In a DPCM system, we know that it is important for the samples not to have large deviations to achieve the best possible performance relative to space cost, as each level will be represented by more bits. We also observe that increasing **p** reduces the quantization error but to a lesser degree compared to **N**. This happens because the prediction uses more samples, making it better up to a certain point.

In conclusion, increasing **N** certainly improves performance, while increasing **p** provides minimal to no improvement.