# 70103 Statistical Information Theory CW

Ali Abbas

November 25, 2024

## 1 Analytic Comparison

The probability of correctly decoding a Hamming code with $n = 2^m - 1$ bits can be decomposed as such:

$$P_{\text{correct}} = P_{\text{no error}} + P_{\text{single bit flip}}$$

The probability of no errors is:

$$P_{\text{no error}} = (1-p)^n$$

The probability of exactly one bit flipping is:

$$P_{\text{single bit flip}} = \binom{n}{1} \cdot p \cdot (1-p)^{n-1}$$

And so:

$$\begin{aligned} P_{\text{correct}} = &\ (1-p)^n + \binom{n}{1} \cdot p \cdot (1-p)^{n-1} \\ = &\ (1-p)^n \cdot \left(1 + \cdot \frac{n \cdot p}{1-p}\right) \end{aligned}$$

Therefore, for a completely noisy ($p = 0.5$) binary symmetric channel, for $m \in \{2, 3, 4\}$ we have:

$m = 2 : P_{\text{correct}} = 0.5$
$m = 3 : P_{\text{correct}} = 0.0625$
$m = 4 : P_{\text{correct}} = 0.00048828125$

And the numerical results are similar:

$m = 2 : P_{\text{correct numerical}} = 0.485$
$m = 3 : P_{\text{correct numerical}} = 0.066$
$m = 4 : P_{\text{correct numerical}} = 0.0$

In the figure below is the code for a hypothesis test on these values:

```python
from scipy import stats
import pandas as pd

# flip probability
p = 0.5

# standard deviation when modelling the binary symmetric channel as a bernoulli trial
sigma = (p * (1 - p)) ** 0.5

n = 1000  # number of trials
SE = sigma / (n ** 0.5)  # standard error of the mean
alpha = 0.05  # significance level
z = stats.norm.ppf(1 - alpha / 2)  # z-value for 95% confidence interval

data = pd.DataFrame({
    "m": [2, 3, 4],
    "P_expected": [0.5, 0.0625, 0.00048828125],
    "P_numerical": [0.485, 0.066, 0.0],
})

# confidence intervals
data["CI_lower"] = data["P_expected"] - z * SE
data["CI_upper"] = data["P_expected"] + z * SE

# Check if numerical values fall within the confidence intervals
data["Within_CI"] = \
  (data["P_numerical"] >= data["CI_lower"]) \
    & \
  (data["P_numerical"] <= data["CI_upper"])
```

Figure 1: Hypothesis Test

And the results indicate that indeed the numerical values are within the expected range:

| | m | P_expected | P_numerical | CI_lower | CI_upper | Within_CI |
|---|---|---|---|---|---|---|
| 0 | 2 | 0.500000 | 0.485 | 0.469010 | 0.530990 | True |
| 1 | 3 | 0.062500 | 0.066 | 0.031510 | 0.093490 | True |
| 2 | 4 | 0.000488 | 0.000 | -0.030501 | 0.031478 | True |

Figure 2: Hypothesis Test Results

## 2 Visual Comparison

In this section there are 3 plots which show the decoder accuracy vs the noise probability for $m \in \{2, 3, 4\}$, as well as a final combined plot to visualise the differences between various $m$.

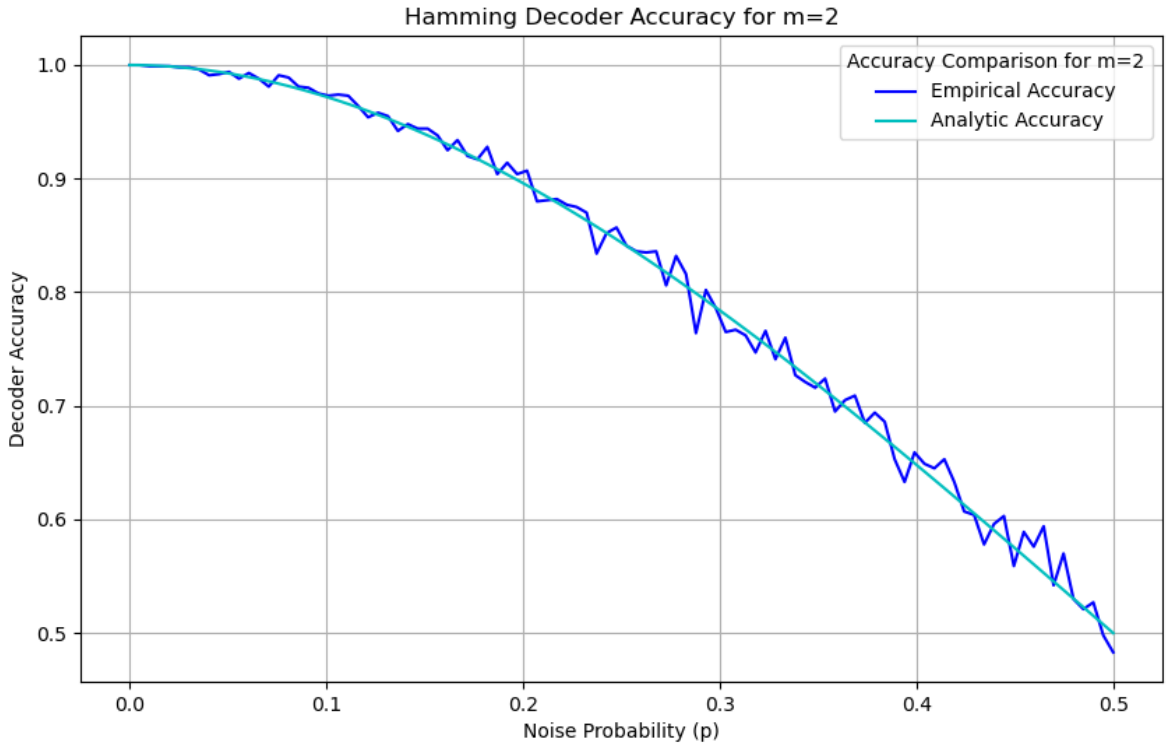


Figure 3: Hamming Decoder Accuracy for $m = 2$

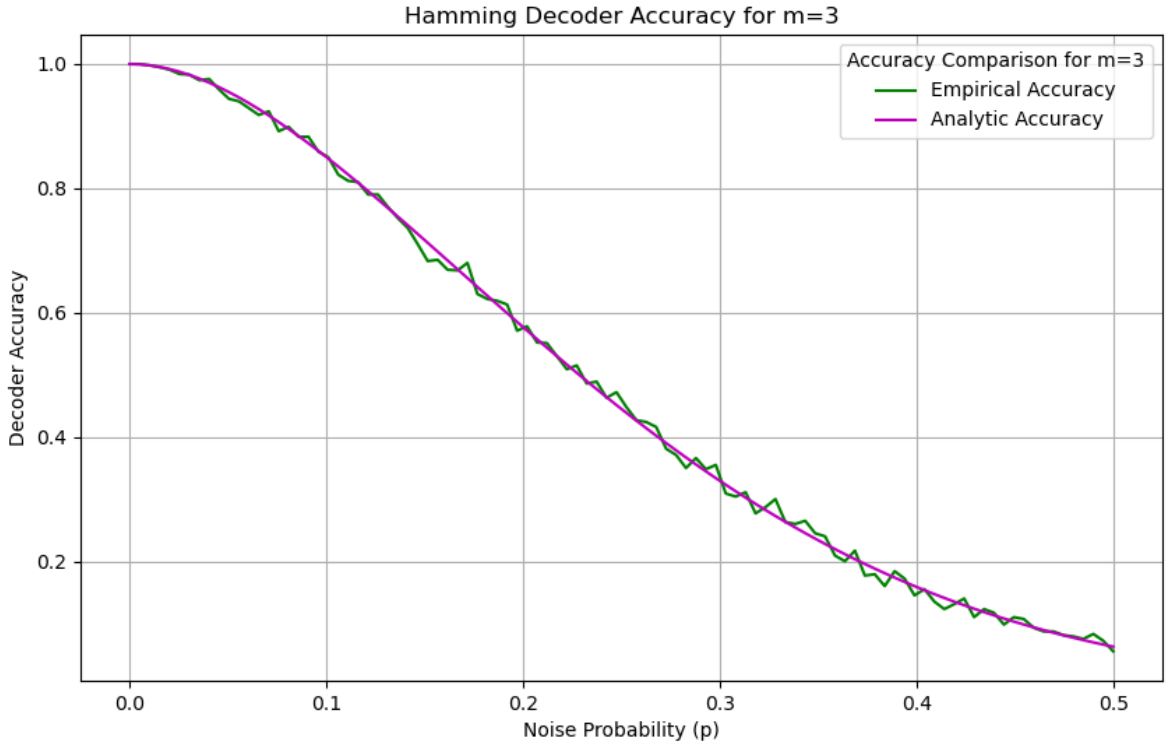


Figure 4: Hamming Decoder Accuracy for $m = 3$

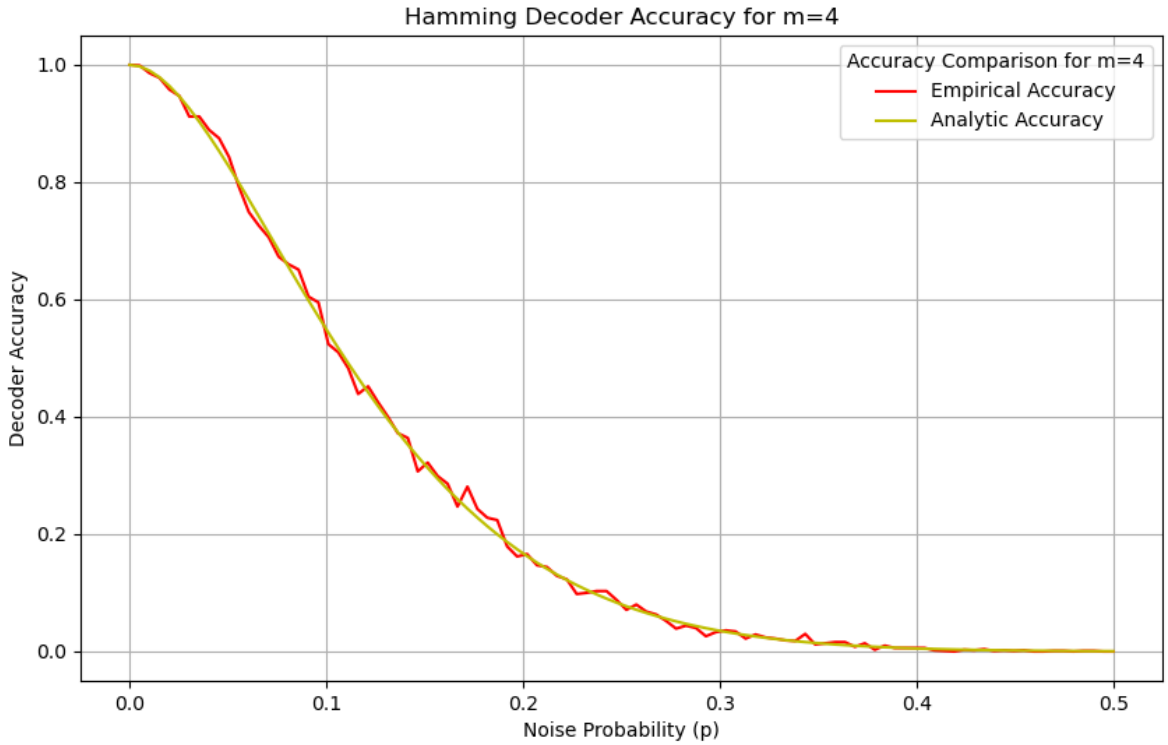


Figure 5: Hamming Decoder Accuracy for $m = 4$

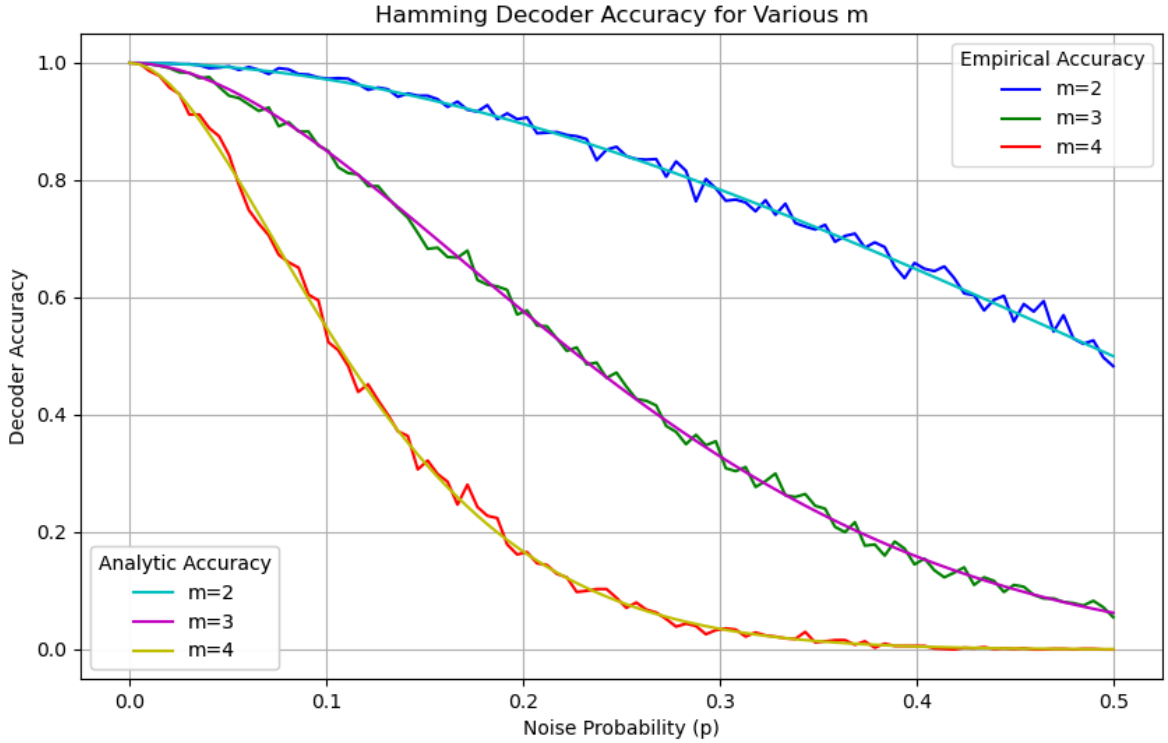


Figure 6: Hamming Decoder Accuracy for $m \in \{2, 3, 4\}$

From these figures, I conclude that Hamming codes perform well for low noise settings, but their error correcting abilities are impacted as noise increases (since Hamming codes can't correct multiple bit errors). For subsequently higher values of $m$ (and thus for longer codewords) the rate of dropoff in performance increases considerably, highlighting that they aren't suitable for correcting multiple bit errors. The dropoff in performance is intuitively due to the decreased proportion of error correcting bits relative to data bits for higher values of $m$, which reduces the redundancy of the code.