BitSync Task - Reed Solomon Codes

Arjun Arunachalam

April 2025

1 Introduction

Reed-Solomon codes were developed by Iriving S Reed and Gustave Solomon when working at MIT Labs in 1960. They use an alphabet set of length $q = p^m$, q being a prime power (i.e. the encoding set is a galois field) and encode the input at n points; later interpolating it to a polynomial. Using this, errors are detected or corrected and **binary erasures** can be taken care of.

2 Documentation

2.1 `SimulateChannel.py`

- Requirements

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

- Function to encode the text. Returns a list of encoded values. In this case, the encoding is just the number assigned to ASCII characters in Python.

```
def encode_alphabet_naive(text):
```

- A custom encoding based on values modulo (3).

```
def encode_alphabet_custom(text):
```

- Function to decode. Here, decoding is based on the original function; i.e. Python assigned ASCII value to character.

```
def decode_alphabet(encoded_values):
```

- Returns a list of random values from the list $[1, n]$, with n being the length of the pdist. The values are chosen based on corresponding probabilities in the pdist.

  ```
  def random_out(pdist):
      return np.random.choice(np.arange(1, len(pdist) + 1), p=pdist)
  ```

  Example: If pdist = [0.1, 0.1, 0.8], output could be [1, 3, 3] as 3 has higher probability.

- Plot the channel matrix as a heatmap.

  ```
  def plot_channel_matrix(probab_matrx):
  ```

- Mean squared error function. Inputs actual, predicted are lists. Converted to numpy array to support index by index operations directly (actual - predicted = array of differences in values).

  ```
  def mean_squared_error(actual, predicted):
      actual = np.array(actual)
      predicted = np.array(predicted)
      return np.mean((actual - predicted) ** 2)
  ```

- Normalize rows and columns of the probability matrix.

  ```
  probab_matrx = probab_matrx / probab_matrx.sum(axis=1, keepdims=True)
  probab_matrx = probab_matrx / probab_matrx.sum(axis=0, keepdims=True)
  ```

- Show results and error of custom encoding and ASCII encoding. To decide optimal encoding.

  ```
  plot_channel_matrix(probab_matrx)
  ```

2.2 `Hamming_Code_Encoder_Decoder.py`

- Requirements. Numpy
- Define Generator matrix G. (4, 7) and parity check matrix H (3, 7).
- Encoder. Comverts 4 bit to 7 bit. Accepts a 4 bit encoding as input. Conversion is done by taking dot product of encoding with a generator matrix.

  ```
  def encode_batch(data_bits):
  ```

- Decoder. Takes in 7-bit input and converts it into the original 4-bit message. Procedure as follows.
  - Get syndrome for error to bit mapping. Take dot product of parity check matrix and transpose of received code.
  - Obtain error bit based on positions obtain in syndrom tuple.
  - Copy the bit to a temporary variable, corrected.
  - If the bit is errored, flip corrected.
  - Append corrected to decoded array.

  ```
  def decode_batch(received_batch):
  ```

- Simulate a channel with gaussian noise. Initialise standard deviation of noise distribution. Array of length = codewords.shape (length of code) picked from the distribution using np.random.normal(). Add noise to the codeword array and return the clipped array (round off values to integral/binary bits) using np.clip and np.round.

  ```
  def gaussian_channel(codewords, noise_std=0.5):
  ```

- Simulate a channel with uniform noise; i.e. bit flipping with probability p. Induce flips with probability p by choosing random integers in the range [1, codewords.shape], with probability flip_prob. Return the error induced array by adding and taking mod 2 for binary. Can be extended to n-nary.

  ```
  def uniform_channel(codewords, flip_prob=0.1):
  ```

- Return the bit error rate = number of errored bits/total number of bits.

  ```
  def bit_error_rate(original, decoded):
  ```

2.3 `Reed_Solomon_Encoder_Decoder.py`

- Requirements. `numpy, galois.`
- Simulate binary erasure channel with erasure_prob. An array of random indices is chosen with probability erasure_prob using np.random.rand. Original codeword is copied to an array and chosen indices are replaced with '?' to indicate erasures. Returns the array.

  ```
  def binary_erasure_channel(codeword, erasure_prob):
  ```

- Class for Reed Solomon Error Correction. Initialised variables and functions.
  - m = Degree of field.
  - GF = Galois Field of order m for the prime $p = 2$ in this case.
  - n = Length of the codeword
  - k = Total length of the message.
  - `def rs_encode(self, message)` - First check for length of message. Fit the message into a polynomial in the GF by using `galois.Poly()` and store its range in a variable xs. The codeword will be the set of values at the points $\{x_i\}$; stored in the variable codeword.
  - `def rs_decode(self, received)` - Decode the message by computing the value of the interpolated polynomial at the points $\{x_i\}$. Use Lagrange Interpolation inbuilt in galois library. xs = $\{x_i\}$ is known and input is the receieved code. Add zeroes in the message to account for erasures and to maintain length of decoded message.
  - `def compute_rate(self)` - Returns rate = k/n. (Number of message bits/Total number of bits).
- `main()` to call functions and display results.