Final Project: The Blahut-Arimoto Algorithm

Student: Veronica Marrocco
Course: ELECENG 4TK4

Accompanying Code: *Python Implementation Link*

Consider an additive Gaussian noise channel **Y = X + Z**, where Z ~ N (0, 1) is independent of X. We use the Blahut-Arimoto algorithm to compute the capacity-achieving distribution of this channel subject to the peak power constraint $|X| \leq P$ for P = 0.1, P = 1, and P = 10. That is, every the power of every symbol in the codeword that is transmitted is bounded by P:

$$x_t^2(m) \leq P \text{ for t = 1, 2…, n and m = 1, 2,..., } 2^k$$

Where n is the length of each codeword, and 2^k is the number of codewords in the codebook.

The Blahut-Arimoto algorithm is an interactive optimization method commonly used to compute the channel capacity of discrete memoryless channels and the corresponding capacity-achieving input distribution. In brief, it alternates between updating the posterior conditional probability Q(x|y) and the input distribution p(x), refining p(x) according to the current Q(x|y) to ensure the mutual information I(X;Y) increases with each iteration. Once successive changes to p(x) become sufficiently small (within some tolerance threshold), it can be inferred that p(x) has converged to the optimal input distribution and that the channel capacity has been reached.

For the additive Gaussian noise channel presented here, which is a continuous channel, the Blahut-Arimoto method was adapted using discretization of input and output alphabets. The algorithm was implemented in python using PyTorch tensors and the Plotly library for graphics. The key functions and their theoretical background are presented below.

Description of Python functions for implementation:

1. Theoretical Capacity: **theoreticalCapacity(P, _sigma=1)**

The theoretical capacity of a Gaussian additive white noise channel is:

$$C = 0.5\, log(1 + \frac{P}{\sigma^2})$$

Where P is the peak power constraint, and σ^2 is the variance of the Gaussian noise, which is assumed to be 1 in this project. The *theoreticalCapacity* function computes the theoretical capacity given P and $\sigma = 1$ in order to verify the algorithm's results.

2. Input Power Calculation: **getInputPower(p_x, x)**

Gaussian variables are continuous, but we discretize them in order to implement the Blahut-Arimoto algorithm. The power that we are constraining at the symbol-level (assuming a mean of zero) is given by:

$$var(x) = E[x^2] - E[X]^2 = \sum_{x \epsilon X} x^2 p(x)$$

The *getInputPower* function computes this input power to ensure the input distribution adheres to the power constraint.

3. Conditional Transition Probabilities: **getChannel(x, y, sigma=1)**

The noisy Gaussian channel can be modelled as a conditional transition matrix, P(y|x). The *getChannel* function computes P(y|x), attributing to each (x, y) pair in the joint alphabet the probability of Y falling within a small range around y. This is done by using the CDF of the Gaussian distribution, and normalization follows to ensure probabilities sum to 1.

4. Blahut-Arimoto Algorithm: **runAlgorithm(<parameters>)**

The algorithm function takes the following parameters as inputs:

- P: Peak power constraint.
- σ: Noise standard deviation (which is just 1 for this case).
- max_iter: Maximum number of iterations.
- NX: Number of discretized points for input alphabet.
- NY: Number of discretized points for output alphabet.
- Tolerance: Convergence criterion (ensuring adherence to power constraint).
- ε: Small value for numerical stability.

The *runAlgorithm* function computes and returns the channel capacity for a given peak power constraint, P. This is the heart of the code, iteratively optimizing the input distribution to maximize mutual information, while ensuring the input power constraint is satisfied.

4.1. Discretization & Initialization

First, we discretize the input and output alphabets. The input alphabet takes uniformly spaced points in [-P, P], respecting the peak power constraint. The output alphabet covers [-P - 4σ, P + 4σ] in order to account for Gaussian noise spread. p(x) is initialized as the uniform distribution.

4.2. Iterative Updates

The Blahut-Arimoto algorithm maximizes I(X;Y) by means of posterior probabilities, Q(x|y), which is the posterior knowledge of X given Y. By updating Q(x|y) based on the current p(x) distribution, the algorithm identifies how effectively each input x contributes to distinguishing y. With each iteration, p(x) is brought closer and closer to the optimal input distribution, until reasonable convergence is achieved.

4.2.1. The output distribution p(y) is computed by marginalizing over x using the current p(x) and the channel matrix p(y|x):

$$p(y) = \sum_{x \epsilon X} p(x, y) = \sum_{x \epsilon X} p(y|x)p(x)$$

4.2.2. The posterior distribution Q(x|y) is computed according to Bayes' rule:

$$Q(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

4.2.3. The input distribution p(x) is refined according to the Q(x|y) found, in order to continuously optimize the mutual information based on the contribution of each x. This is achieved as follows.

Mutual information is defined as:

$$I(X;Y) = \sum_{(x,y)\epsilon XxY} p(x,y) log \frac{p(x,y)}{p(x)p(y)}$$

Rewriting in terms of the channel matrix, P(y|x):

$$I(X;Y) = \sum_{(x,y)\epsilon XxY} p(x)p(y|x) log \frac{p(y|x)}{p(y)}$$

$$I(X;Y) = \sum_{x\epsilon X} p(x) \sum_{y\epsilon Y} p(y|x) log \frac{p(y|x)}{p(y)}$$

By Bayes' rule:

$$Q(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Therefore by taking log of either side and multiplying by P(y|x):

$$P(y|x)\ logQ(x|y) = P(y|x)\ log \frac{p(y|x)p(x)}{p(y)}$$

$$P(y|x)\ logQ(x|y) = P(y|x)\ log \frac{p(y|x)}{p(y)} + P(y|x) logp(x)$$

$$P(y|x)\ log \frac{p(y|x)}{p(y)} = P(y|x)\ logQ(x|y) - P(y|x) logp(x)$$

Substituting this into the expression for mutual information:

$$I(X;Y) = \sum_{x\epsilon X} p(x) \sum_{y\epsilon Y} P(y|x)\ logQ(x|y) - P(y|x) logp(x)\ (\mathbf{a})$$

Therefore, p(x) is iteratively optimized by assigning higher probabilities to x-values that contribute most to maximizing I(X;Y). That is, in accordance with the relationship established in (**a**):

$$p(x)\ \alpha\ exp(\sum_{y\epsilon Y} P(y|x)\ logQ(x|y))$$

4.2.4. Finally, the difference between successive input distributions p(x) is computed. Once they are below a specified tolerance level, the algorithm stops. This ensures that the updates to p(x) have become small enough that the algorithm can stop iterating, because it is assumed that the distribution has converged to the optimal one.

5. Plotly Code

This implementation uses the Plotly library to create dynamic plots that visualize the results of the Blahut-Arimoto algorithm for the Gaussian noise channel. The plots include the input probability distributions p(x) and corresponding output probability distributions p(y) for varying peak power constraints. Iterative changes in p(x) are also visualized dynamically and statically, showing how the distribution converges to its final shape during the algorithm's iterations.

Key Results:

The functions described above were called for the three specified power constraints, being P = 0.1, P = 1 and P = 10. The following parameters were used:

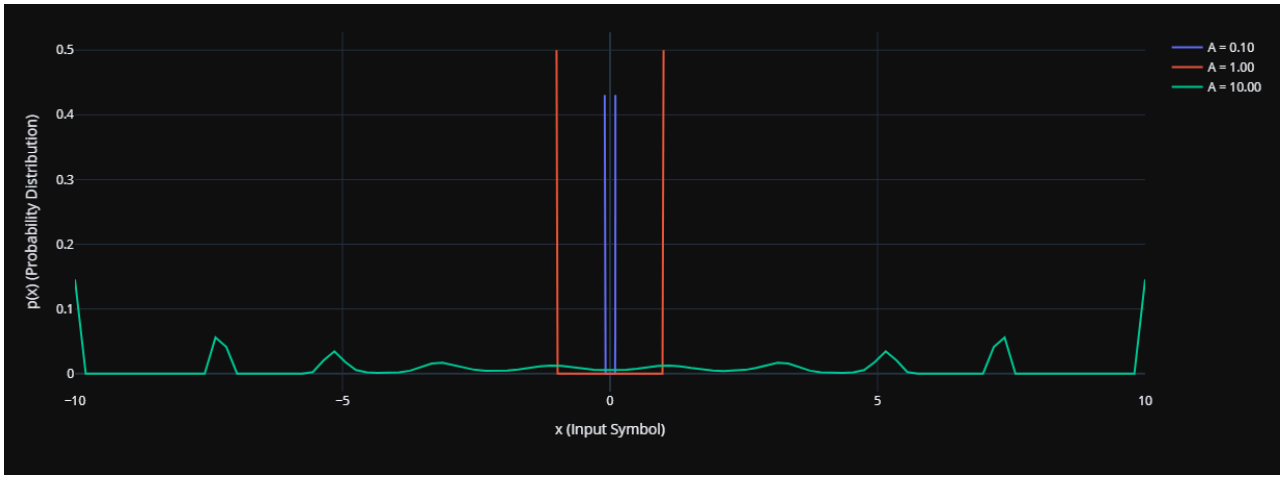
- NX = 1000 (number of discretized points in input alphabet)

- NY = 2000 (number of discretized points in output alphabet)
- Tolerance for convergence = 10^{-5}

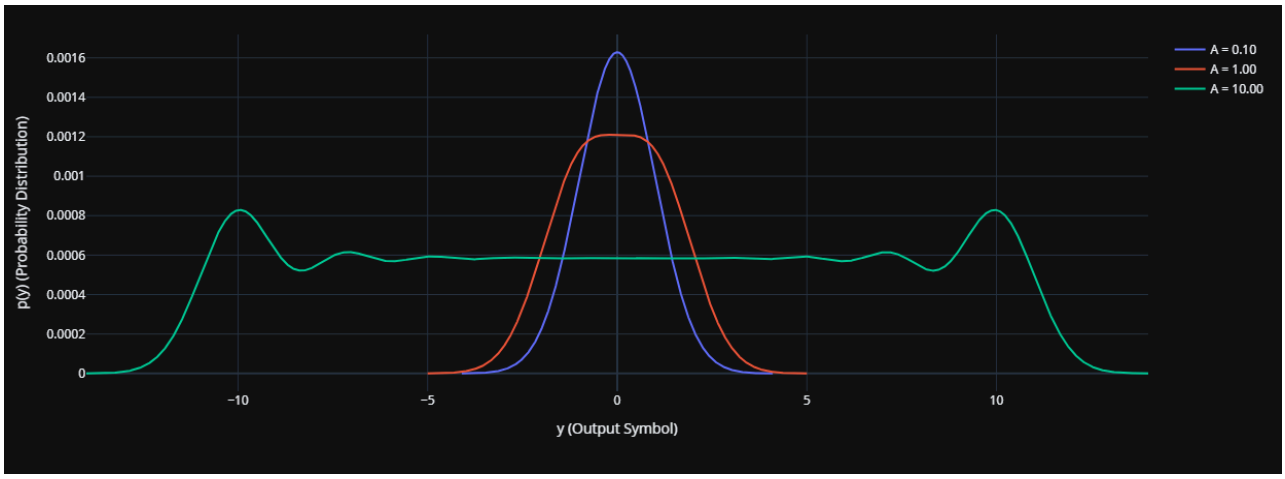
Results Summary Table:

Power Constraint	P = 0.1	P = 1	P = 10
Total Iterations for Convergence	10,000	925	10,000
Theoretical Capacity	0.069	0.500	1.730
Measured Channel Capacity	0.005	0.337	1.758

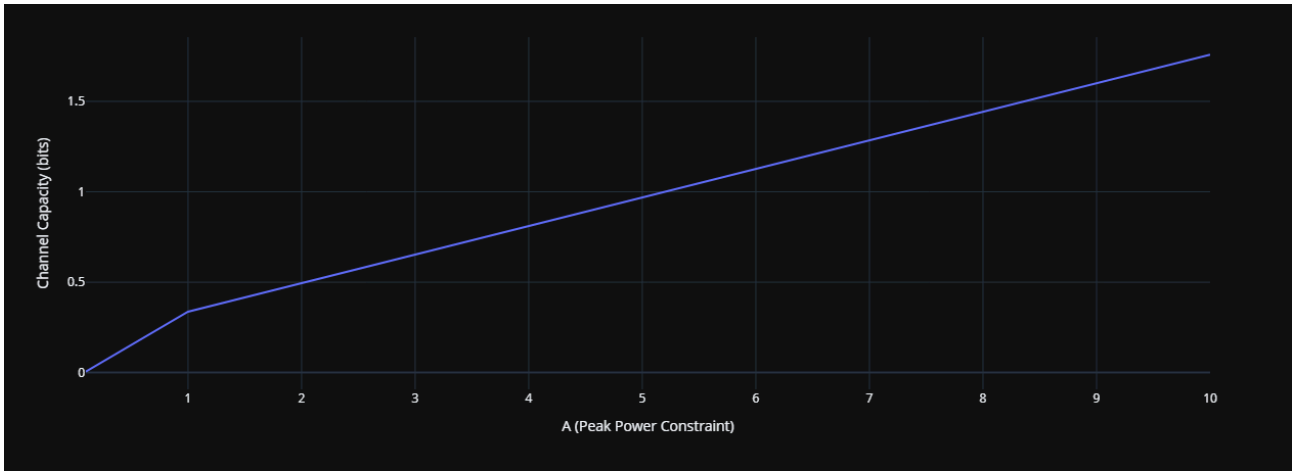
Input distributions p(x) for the different values of P:



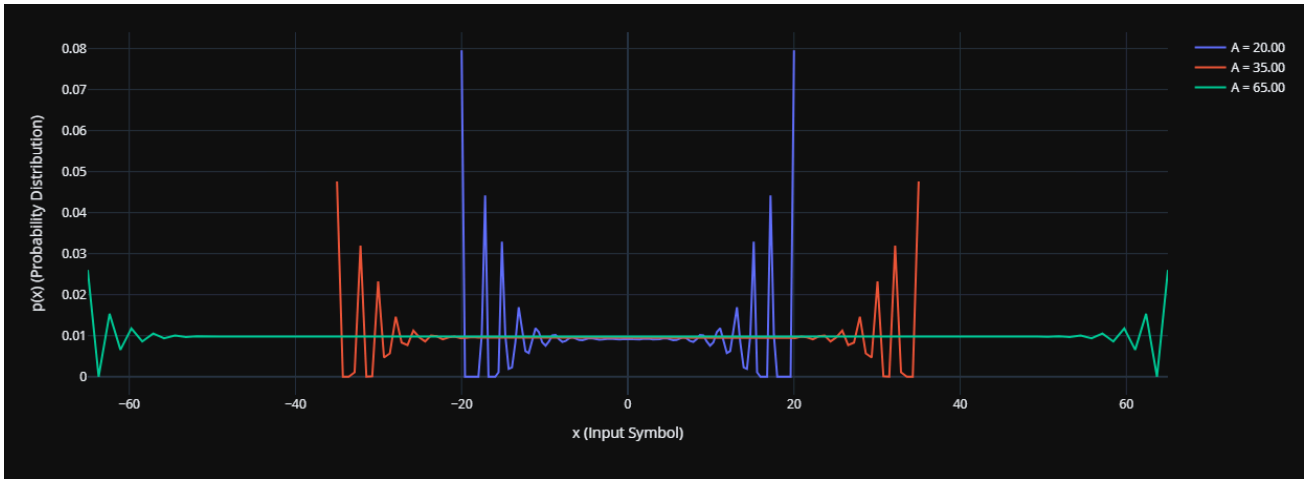
Output distributions p(y) for the different values of P:



Channel Capacity vs. Peak Power Constraint:



Additional input distributions for larger P values (P = 35 and P = 65)



Additional output distributions for larger P values (P = 35 and P = 65)

Observations:

1. **Input & Output Distributions**

For all optimal input distributions (regardless of P), the probability density saturates at the extreme values, i.e. ±0.1 for P = 0.1, ±1 for P = 1 and ±10 for P = 10. This is because greater power, or greater input variance, reduces the probability of error for a given rate, thereby increasing channel capacity. When the variance is maximized, the input symbols are furthest apart, meaning the channel can generate output symbols with Gaussian distributions that are also furthest apart. The decreased overlap in the output (Y) space results in a lower probability of error. Consequently, as is shown in the results, the optimal input distributions maximize variance by concentrating at the extreme values ±P.

For P = 10, we observe additional smaller peaks within the range [-10, 10], aside from the dominant peaks ±10. This finding has a geometric interpretation. Recall the definition of mutual information in terms of entropy:

$$I(X;Y) = H(Y) - H(Y|X)$$

I(X;Y) is therefore maximized when the entropy of the output distribution, H(Y), is also maximized. This occurs when p(y) is as "wide" as possible, approaching the uniform distribution, while maintaining distinguishable symbols. If the input distribution only concentrates at ±10, some areas of the output range, particularly near y = 0, would be underutilized. This would decrease H(Y) and fail to maximize the channel's dynamic range.

For the smaller values of P (0.1 and 1), we only observe the large peaks at ±P, with no significant probability mass elsewhere. This is because, at smaller P, the signal range of [-P, +P] is closer to the noise level ($\sigma^2 = 1$), and hence maximizing the input variance takes on the utmost importance. Otherwise, the noise begins to trump the input signal, and the mutual information between X and Y drops drastically.

For even larger values of P (e.g., P = 35 and P = 65), a similar trend is observed. As P increases, and the system is less constrained, the optimal input distribution becomes less and less saturated at the extreme values, adopting a more and more "Gaussian" shape. This is consistent with the fact that, for an unconstrained additive Gaussian noise channel, the optimal input distribution is also Gaussian. Simultaneously, with higher P values, the output distributions appear to broaden, maximizing the use of the available range and increasing H(Y).

2. **Capacity Trends**

It was found that the channel capacity increases with P, since higher P allows for larger input variance, which the Gaussian channel leverages for improved data transmission. Greater "distance" can be allowed between input symbols, which leads to less overlap in the output Gaussian distributions centered at Y = x. Hence, the mutual information between X and Y is greater, increasing C by definition.