Outline

1. Belief Propagation
   - Sum-Product Algorithm
   - BEC
   - BI-AWGNC
   - MATLAB Implementation

2. Density Evolution
   - BEC
   - BI-AWGNC
   - Code Optimization

3. EXIT Charts
   - BEC
   - BI-AWGNC
   - Code Optimization
Sum-Product Algorithm

- Message passing algorithm.
- Used for efficient marginalization of factorizable functions which can be represented as factor graphs.
- Decoding of LDPC Codes can be expressed as such a function.
- Optimal on graphs with no cycles, generally suboptimal.
- Polynomial time complexity with respect to code length.
Consider the function

\[ f(x_1, x_2, x_3, x_4, x_5, x_6) = f_1(x_1, x_2, x_3)f_2(x_1, x_4, x_6)f_3(x_4)f_4(x_4, x_5) \]

For the marginalization wrt \( x_1 \), we can write

\[ f(x_1) = \sum_{x_2, \ldots, x_6} f(x_1, \ldots, x_6) \]

\[ = \sum_{x_1} f_1(x_1, x_2, x_3)f_2(x_1, x_4, x_6)f_3(x_4)f_4(x_4, x_5) \]

\[ = \left[ \sum_{x_2, x_3} f_1(x_1, x_2, x_3) \right] \left[ \sum_{x_4, x_5, x_6} f_2(x_1, x_4, x_6)f_3(x_4)f_4(x_4, x_5) \right], \]

simplifying the summation.

The above can be applied recursively.
Bit-wise MAP decoding of LDPC codes can be written as follows

\[ \hat{x}_i^{\text{MAP}}(y) = \arg \max_{x_i \in \{\pm 1\}} p(x_i | y) \]

\[ = \arg \max_{x_i \in \{\pm 1\}} \sum_{\sim x_i} p(x | y) \]

\[ = \arg \max_{x_i \in \{\pm 1\}} \sum_{\sim x_i} p(y | x)p(x) \]

\[ = \arg \max_{x_i \in \{\pm 1\}} \sum_{\sim x_i} \prod_{j=1}^n p(y_i | x_i)[x \in C], \]

where

\[ [x \in C] = \begin{cases} \frac{1}{\lvert C \rvert}, & x \in C, \\ 0, & \text{otherwise}. \end{cases} \]
Figure: Factor graph of an LDPC code.
The Sum-Product Algorithm - Processing Rules

\[ \mu(x) = \prod_{k=1}^{K} \mu_k(x) \]

\[ \mu(x) = \sum_{x} f(x_1, x_2, \ldots, x_J) \prod_{j=1}^{J} \mu_j(x_j) \]

**Figure:** Processing rules for variable (left) and check (right) nodes.
The Sum-Product algorithm is usually called Belief Propagation when the messages represent beliefs.

For the BEC, BP boils down to the following very simple rules.

- At variable nodes, if all of the messages used for the computation of the outgoing message are erasures, then the outgoing message is an erasure.
- At check nodes, if at least one of the messages used for the computation of the outgoing message is an erasure, then the outgoing message is an erasure.
For other binary channels, things are a bit more complicated.

Log-domain decoder is used due to simplification of rules and numerical stability of the log function.

LLRs are used as messages, i.e. each variable node calculates

\[
l = \log \left( \frac{p(y_i|x_i = +1)}{p(y_i|x_i = -1)} \right)
\]

If \( l > 0 \), then \( x_i = +1 \), if \( l < 0 \) then \( x_i = -1 \).

\( l \neq 0 \) with probability 1, so we don’t care about the equality.

In practical scenarios, 0 can be assigned to any value of \( x_i \) or the decision can be randomized with no perceivable effect on the results.
At variable nodes, due to the log function, the outgoing message is calculated as the sum of the corresponding incoming messages.

At check nodes, the outgoing message is calculated according to the following equation

$$m_{cv_{ji}} = 2 \tanh^{-1} \left( \prod_{k \neq i} \tanh \left( \frac{m_{vc_{kj}}}{2} \right) \right),$$

where $m_{cv_{ji}}$ represents the message sent from check node $j$ to variable node $i$ and $m_{vc_{ij}}$ represents the message sent from variable node $i$ to check node $j$. 
Belief Propagation - MATLAB Implementation

Note: the following implementation of belief propagation for the AWGN channel aims at readability and intuitiveness, *not* at speed.

Data structures used:
- $m \times n$ matrix $H$.
- $m \times n$ matrix variable_to_check.
- $m \times n$ matrix check_to_variable.
- $1 \times n$ matrix for initial LLRs.
- $1 \times n$ matrix $c_{\text{hat}}$ for the intermediate hard decisions.
Belief Propagation - MATLAB Implementation

Algorithm:

1. Initialize $1 \times n$ matrix for LLRs based on channel observations.
2. For each variable node (i.e. for $i = 1 : n$) calculate the outgoing message to check node $j$ based on the values in $\text{check}\_\text{to}\_\text{variable}(::, i)$, where $j$ are the indices of all non-zero entries in $H(:, i)$ and store it in $\text{variable}\_\text{to}\_\text{check}(j, i)$.
3. For each variable node also calculate the overall LLR and make a hard decision for each bit based on the sign of the LLR.
4. Calculate $\hat{c} \cdot H^T$. If it is equal to an all-zero $1 \times m$ vector, decoding is successful. Else, continue.
5. For each check node (i.e. for $i = 1 : m$) calculate the outgoing message to variable node $j$ based on the values in $\text{variable}\_\text{to}\_\text{check}(i, :)$, where $j$ are the indices of all non-zero entries in $H(i, :)$ and store it in $\text{check}\_\text{to}\_\text{variable}(i, j)$.
6. If maximum number of iterations has not been reached, go to step 2, else declare failure.
In the limit of infinite blocklength and assuming that the all-zero codeword is transmitted, the performance of an ensemble of LDPC codes can be predicted by Density Evolution.

Density Evolution tracks the evolution of message pdfs throughout the decoding procedure.

For the BEC, a single parameter suffices to track this density, i.e. the average bit erasure probability.

For the BI-AWGNC, whole densities have to be tracked as the received LLR messages have a continuous pdf.
Recall that the degree distribution of a code from an edge perspective is denoted as follows:

\[ \lambda(x) = \sum_i \lambda_i x^{i-1} \quad \text{and} \quad \rho(x) = \sum_i \rho_i x^{i-1} \]

Also recall that the degree distribution from a node perspective associated with \( \lambda(x) \) is denoted as \( L(x) \).

For the BEC(\( \epsilon \)), Density Evolution takes on the following simple form:

\[ x_\ell = \epsilon \lambda(1 - \rho(1 - x_{\ell-1})) \]
\[ P^b_\ell = \epsilon L(1 - \rho(1 - x_{\ell-1})) \]

where \( x_0 = \epsilon \) and \( P^b_\ell \) is the bit erasure probability at iteration \( \ell \).
In order for the probability of erasure to converge to zero, the following has to hold for each $\ell$

$$x_\ell > x_{\ell+1} = \epsilon \lambda (1 - \rho (1 - x_\ell))$$

**Figure:** Density evolution for a (3, 6) regular LDPC code.
Density Evolution - BI-AWGNC

- DE for the BI-AWGNC tracks the pdf of LLR messages transmitted across the graph edges throughout the decoding procedure.
- At a degree $i$ variable node, the output density is the $(i - 1)$-fold convolution of the input density, which is then convoluted with the channel LLR density.
- As a degree $i$ check node, the output density is the $(i - 1)$-fold convolution of the input density after a transformation into the so-called $G$-domain in which convolution is defined in a slightly different way.
- Computationally demanding, not suitable for optimization procedures.
- Gaussian Approximation (GA) assumes that all distributions are symmetric Gaussian (i.e. variance $\sigma^2$ and mean $\sigma^2/2$).
- Only one parameter needs to be tracked, the mean.
At a variable node of degree $i$, the mean of the outgoing message $\mu_{v_i}$ is $(i - 1)$ times the mean of the incoming messages plus the mean of the channel message.

Due to the central limit theorem, the Gaussian assumption is a good approximation for the variable node messages.

For the check nodes, we first need to define the function $\phi(x)$

$$
\phi(x) = \begin{cases} 
1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{+\infty} \tanh \left( \frac{u}{2} \right) \exp \left\{ -\frac{(u-x)^2}{4x} \right\} du, & x > 0 \\
1, & x = 0.
\end{cases}
$$

$\phi(x)$ can either be approximated by another function or precalculated.
At a check node of degree $i$, the mean of the outgoing message can be calculated as follows.

$$\mu_{u,i}^{(\ell)} = \phi^{-1} \left(1 - \left[1 - \sum_{i} \lambda_i \phi \left(\mu_{v_i}^{(\ell-1)}\right)\right]^{j-1}\right)$$

Averaging over $\rho(x)$, we will have the overall density $\mu_{u}^{(\ell)}$.

In order for decoding to converge to a vanishingly small probability of error, the following has to hold for all $\ell$

$$\mu_{u}^{(\ell)} > \mu_{u}^{(\ell-1)}.$$
The usual approach is to maximize the rate for given noise variance (or erasure probability), while ensuring a small probability of error. The rate of a code can be expressed as

\[ r = 1 - \frac{\sum_i \rho_i / i}{\sum_i \lambda_i / i}. \]

By fixing \( \rho(x) \), the objective function becomes linear, i.e., maximization of \( \sum_i \lambda_i / i \).

The optimization problems resulting from Density Evolution are not linear, so they cannot be solved optimally with reasonable complexity.

An efficient genetic algorithm, called Differential Evolution, is commonly used.
For the BEC, the DE convergence condition can be made slightly more strict as follows

$$\epsilon \lambda (1 - \rho(1 - x)) < x, \quad \forall x \in (0, 1).$$

In fact, it suffices if the above holds in $(0, \epsilon]$. Now, for fixed $\rho(x)$, the problem is linear and can be solved efficiently. With the same approach, DE with GA can be rewritten as a linear program.
Using the linear programming approach for $\epsilon = 0.5$ and $\nu_{\text{max}} = 15$, we were able to find a code with

$$\lambda(x) = 0.4001x + 0.1441x^2 + 0.1703x^3 + 0.0734x^7 + 0.2122x^8$$

and $\rho(x) = x^5$, which has rate $r = 0.4846$.

Figure: Density Evolution for the code found above.
If we define $\nu_\epsilon(x) = \epsilon \lambda(x)$ and $c(x) = 1 - \rho(1 - x)$, then the stricter DE recursion can be written as

$$\nu_\epsilon(c(x)) < x, \quad \forall x \in (0, 1).$$

Since $\nu_\epsilon(x)$ is a polynomial with positive coefficients, it has an inverse and we can write

$$\nu_\epsilon(x)^{-1} > c(x), \quad \forall x \in (0, 1).$$
Figure: EXIT chart for a (3, 6) regular code over a BEC with $\epsilon = 0.43$. 
For the BI-AWGN channel, mutual information between the all-zero codeword $X$ and the LLR corresponding to the Gaussian RV $Y = X + V$, $V \sim \mathcal{N}(0, \sigma^2)$ is usually tracked.

First, let us define the function $J(\sigma)$ as follows

$$J(\sigma) = 1 - \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{+\infty} e^{-\frac{(x-\sigma^2/2)^2}{2\sigma^2}} \log_2(1 + e^{-x}) \, dx.$$ 

$J(\sigma)$ can either be approximated by another function or precalculated.
Using $J(\sigma)$, the EXIT chart of an irregular code $I_{EC}$ describing the variable node function can be computed as follows.

\[
I_{EV}^i(I_{AV}, \sigma_{ch}^2) = J\left(\sqrt{(i - 1)J^{-1}(I_{AV})^2 + \sigma_{ch}^2}\right)
\]

\[
I_{EV}(I_{AV}, \sigma_{ch}^2) = \sum_i \lambda_i I_{EV}^i(I_{AV}, \sigma_{ch}^2),
\]

where $i$ is the variable node degree, $I_{AV}$ is the mutual information of the message entering the variable node with the transmitted codeword $X$, $I_{EV}^i(I_{AV}, \sigma_{ch}^2)$ is called an elementary EXIT chart, $I_{EV}(I_{AV}, \sigma_{ch}^2)$ is the overall EXIT chart.

For $\sigma_{ch}^2$ we have:

\[
\sigma_{ch}^2 = \frac{4}{\sigma_{w}^2},
\]

where $\sigma_{w}^2$ is the variance of the additive white Gaussian noise.
Using $J(\sigma)$, the EXIT chart of an irregular code $I_{EC}$ describing the check node function can be very well approximated as follows.

$$I_{EC}^i(I_{AC}) \approx 1 - J \left( \sqrt{(i - 1)J^{-1}}(1 - I_{AC}) \right)$$

$$I_{EC}(I_{AC}) \approx \sum_i \rho_i I_{EC}^i(I_{AC}),$$

where $i$ is the check node degree, $I_{AC}$ is the mutual information of the message entering the check node with the transmitted codeword $X$, $I_{EC}^i(I_{AC})$ is called an elementary EXIT chart, and $I_{EC}(I_{AC})$ is the overall EXIT chart.
In order for the decoding to converge to a vanishingly small probability of error, the EXIT chart of the variable nodes has to lie above the inverse of the EXIT chart for the check nodes.

As with DE, the target usually is to maximize the rate for a given noise variance or channel erasure probability.

The usual approach again is to fix $\rho(x)$ and optimize only $\lambda(x)$.

This optimization is optimistic for the BI-AWGNC because the variable (check) EXIT chart curve with a GA is slightly higher (lower) than the actual curve, so the resulting codes may have a slightly lower threshold than the one predicted.
We set the maximum variable degree to 100 and the channel noise variance to \( \sigma^2 = 0.97869 \) which is exactly the Shannon limit for rate-1/2 binary codes.

Using an EXIT chart technique we were able to find the following degree distributions

\[
\lambda(x) = 0.1350x + 0.2816x^2 + 0.2576x^8 + 0.0867x^{33} \\
+ 0.1204x^{34} + 0.0447x^{91} + 0.0740x^{92}
\]

\[
\rho(x) = x^{10}
\]

with associated rate \( r = 0.49303 \).

By increasing \( \nu_{\text{max}} \), we can improve performance. For example, for \( \nu_{\text{max}} = 200 \) we found a code with rate \( r = 0.49670 \).
Figure: EXIT chart of the code with $v_{\text{max}} = 100$. 
