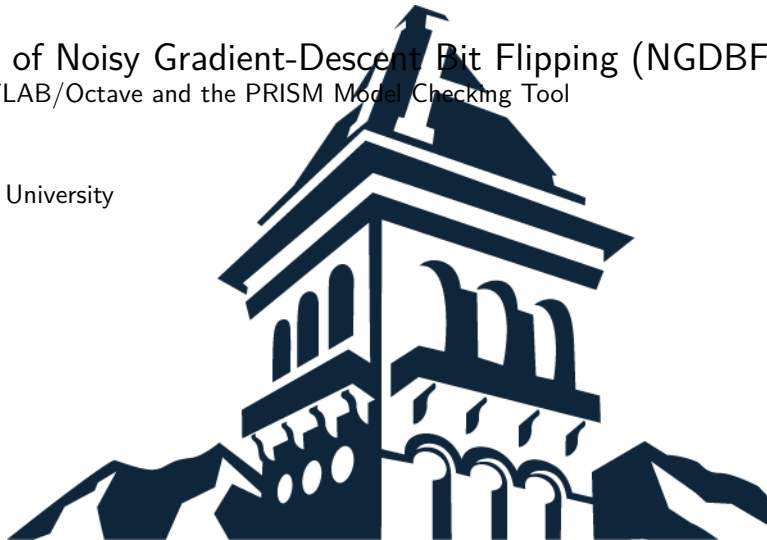


# Analysis of Noisy Gradient-Descent Bit Flipping (NGDBF)

Using MATLAB/Octave and the PRISM Model Checking Tool

Eric Reiss  
Utah State University



# Overview

- ▶ LDPC Codes and Trapping Sets

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview
- ▶ Sample Generation

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview
- ▶ Sample Generation
- ▶ Energy Calculation

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview
- ▶ Sample Generation
- ▶ Energy Calculation
- ▶ Transition Probabilities



# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview
- ▶ Sample Generation
- ▶ Energy Calculation
- ▶ Transition Probabilities
- ▶ Write Files and Process Output

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview
- ▶ Sample Generation
- ▶ Energy Calculation
- ▶ Transition Probabilities
- ▶ Write Files and Process Output
- ▶ What's Next

# Overview

- ▶ LDPC Codes and Trapping Sets
- ▶ Algorithm Overview
- ▶ Model Construction
- ▶ MATLAB/Octave Tool Overview
- ▶ Sample Generation
- ▶ Energy Calculation
- ▶ Transition Probabilities
- ▶ Write Files and Process Output
- ▶ What's Next
- ▶ Sources

# LDPC Codes and Trapping Sets

- Low-Density Parity Check (LDPC) codes were introduced by Gallager in 1963

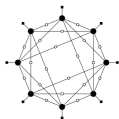


Figure 1:  $(8,8)$  Absorbing set that is dominant in the 802.3an 10GBASE-T LDPC Code [2].

# LDPC Codes and Trapping Sets

- ▶ Low-Density Parity Check (LDPC) codes were introduced by Gallager in 1963
- ▶ LDPC codes are commonly represented as sparse Tanner graph

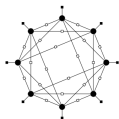


Figure 1: (8,8) Absorbing set that is dominant in the 802.3an 10GBASE-T LDPC Code [2].

# LDPC Codes and Trapping Sets

- ▶ Low-Density Parity Check (LDPC) codes were introduced by Gallager in 1963
- ▶ LDPC codes are commonly represented as sparse Tanner graph
- ▶ Trapping sets are a sub-set of the graph that limit the performance of decoding algorithms, creating an error-floor

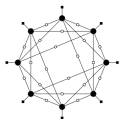


Figure 1: (8,8) Absorbing set that is dominant in the 802.3an 10GBASE-T LDPC Code [2].

# LDPC Codes and Trapping Sets

- ▶ Low-Density Parity Check (LDPC) codes were introduced by Gallager in 1963
- ▶ LDPC codes are commonly represented as sparse Tanner graph
- ▶ Trapping sets are a sub-set of the graph that limit the performance of decoding algorithms, creating an error-floor
- ▶ Absorbing sets are a special case of a trapping sets that are stable in a bit flipping decoder [1]

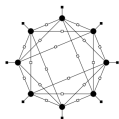


Figure 1: (8,8) Absorbing set that is dominant in the 802.3an 10GBASE-T LDPC Code [2].

# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms



# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]



# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima

# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima
- ▶ Algorithm steps [4]:



# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima
- ▶ Algorithm steps [4]:

▶ Let  $H$  be an  $n \times m$  parity check matrix,  $N_i$  be the adjacency for bit  $i$ ,  $M_j$  be the adjacency for parity check  $j$

# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima
- ▶ Algorithm steps [4]:

- ▶ Let  $H$  be an  $n \times m$  parity check matrix,  $N_i$  be the adjacency for bit  $i$ ,  $M_j$  be the adjacency for parity check  $j$
- ▶ Given channel samples,  $\vec{y}$ , initialize  $\vec{x}$  to be the  $\text{sign}(\vec{y})$

# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima
- ▶ Algorithm steps [4]:

- ▶ Let  $H$  be an  $n \times m$  parity check matrix,  $N_i$  be the adjacency for bit  $i$ ,  $M_j$  be the adjacency for parity check  $j$
- ▶ Given channel samples,  $\vec{y}$ , initialize  $\vec{x}$  to be the  $\text{sign}(\vec{y})$
- ▶ Compute the syndrome,  $s_j = \prod_{i \in M_j} x_i$

# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima
- ▶ Algorithm steps [4]:

- ▶ Let  $H$  be an  $n \times m$  parity check matrix,  $N_i$  be the adjacency for bit  $i$ ,  $M_j$  be the adjacency for parity check  $j$
- ▶ Given channel samples,  $\vec{y}$ , initialize  $\vec{x}$  to be the  $\text{sign}(\vec{y})$
- ▶ Compute the syndrome,  $s_j = \prod_{i \in M_j} x_i$
- ▶ Calculate the energy for bit  $i$ ,  $E_i = y_i x_i + w \sum_{j \in N_i} s_j + z_i$ , where  $w$  is i.i.d white noise and  $z_i$  is a zero mean noise pertubation



# Algorithm Overview

- ▶ NGDBF is part of a family of bit flipping decoding algorithms
- ▶ Improves upon the Gradient-Descent Bit Flipping (GDBF) Decoding Algorithm proposed by Wadayama et al. [3]

▶ GDBF gets “stuck” in local minima while decoding

- ▶ NGDBF introduces psuedo-random noise to escape local minima
- ▶ Algorithm steps [4]:

- ▶ Let  $H$  be an  $n \times m$  parity check matrix,  $N_i$  be the adjacency for bit  $i$ ,  $M_j$  be the adjacency for parity check  $j$
- ▶ Given channel samples,  $\vec{y}$ , initialize  $\vec{x}$  to be the  $\text{sign}(\vec{y})$
- ▶ Compute the syndrome,  $s_j = \prod_{i \in M_j} x_i$
- ▶ Calculate the energy for bit  $i$ ,  $E_i = y_i x_i + w \sum_{j \in N_i} s_j + z_i$ , where  $w$  is i.i.d white noise and  $z_i$  is a zero mean noise perturbation
- ▶ Given a threshold  $\theta$  flip bit  $i$  if  $E_i < \theta$

# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]

# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$



# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$

- ▶ For example, the (3,3) trapping set would have 8 states, 000 - 111

# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$

- ▶ For example, the (3,3) trapping set would have 8 states, 000 - 111
- ▶ A transition from 010 to 000 implies that the middle bit flipped

# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$

- ▶ For example, the (3,3) trapping set would have 8 states, 000 - 111
- ▶ A transition from 010 to 000 implies that the middle bit flipped

- ▶ Edge probabilities are calculated by first calculating the probability that a bit flips using the cdf centered on the energy function with the noise perturbation subtracted

# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$

- ▶ For example, the (3,3) trapping set would have 8 states, 000 - 111
- ▶ A transition from 010 to 000 implies that the middle bit flipped

- ▶ Edge probabilities are calculated by first calculating the probability that a bit flips using the cdf centered on the energy function with the noise perturbation subtracted

- ▶  $P_{flip} = \text{normcdf}(\theta, E_i, \sigma)$

# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$

- ▶ For example, the (3,3) trapping set would have 8 states, 000 - 111
- ▶ A transition from 010 to 000 implies that the middle bit flipped

- ▶ Edge probabilities are calculated by first calculating the probability that a bit flips using the cdf centered on the energy function with the noise perturbation subtracted

$$\text{▶ } P_{\text{flip}} = \text{normcdf}(\theta, E_i, \sigma)$$

- ▶ The probability of a transition is then calculated by multiplying the flip probabilities



# Model Construction

- ▶ The Markov Chain structure used in the tool was proposed in [1]
- ▶ For a given (a,b) trapping set, the state space is described by the a-bit binary representation of the numbers from 0 to  $2^a - 1$

- ▶ For example, the (3,3) trapping set would have 8 states, 000 - 111
- ▶ A transition from 010 to 000 implies that the middle bit flipped

- ▶ Edge probabilities are calculated by first calculating the probability that a bit flips using the cdf centered on the energy function with the noise perturbation subtracted

$$\text{▶ } P_{\text{flip}} = \text{normcdf}(\theta, E_i, \sigma)$$

- ▶ The probability of a transition is then calculated by multiplying the flip probabilities

# MATLAB/Octave Tool Overview

- ▶ Located in the `USU_stochastic_case_studies` repository in the `ngdbf_models` folders, contains the following files



# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files
  - ▶ run\_ngdbf.m: Driver function that contains most of the relevant code

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files
  - ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
  - ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files
  - ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
  - ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
  - ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
- ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed



# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files
  - ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
  - ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
  - ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
  - ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
  - ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
  - ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed
- ▶ The tool uses channel information to programmatically generate a model of the algorithm for each possible initial state

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
- ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed

- ▶ The tool uses channel information to programmatically generate a model of the algorithm for each possible initial state
- ▶ Four main sections of the run\_ngdbf.m are

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
- ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed

- ▶ The tool uses channel information to programmatically generate a model of the algorithm for each possible initial state
- ▶ Four main sections of the run\_ngdbf.m are

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
- ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed

- ▶ The tool uses channel information to programmatically generate a model of the algorithm for each possible initial state
- ▶ Four main sections of the run\_ngdbf.m are

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
- ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed

- ▶ The tool uses channel information to programmatically generate a model of the algorithm for each possible initial state
- ▶ Four main sections of the run\_ngdbf.m are

# MATLAB/Octave Tool Overview

- ▶ Located in the USU\_stochastic\_case\_studies repository in the ngdbf\_models folders, contains the following files

- ▶ run\_ngdbf.m: Driver function that contains most of the relevant code
- ▶ load\_trapping\_sets.m: Script containing some of the trapping sets in University of Arizona's Trapping Set Ontology [5].
- ▶ write\_model.m: Helper function that takes data from run\_ngdbf to write a PRISM model
- ▶ write\_explicit\_model.m: Helper function that creates the necessary files to generate an explicit PRISM model
- ▶ isOctave.m: Helper function to determine if the script is running on Octave or MATLAB
- ▶ get\_error\_sample\_size.m: Helper function to calculate the number of error samples needed

- ▶ The tool uses channel information to programmatically generate a model of the algorithm for each possible initial state
- ▶ Four main sections of the run\_ngdbf.m are

# Sample Generation

- ▶ Samples are pulled from Gaussian distribution with a mean of 1 and a standard deviation of  $\sigma = \sqrt{\frac{1}{R \cdot 10^{SNR/10}}}$ , where  $R$  is the code rate

```
loop_check = get_error_sample_size(sym_size);
valid_samples = zeros(1,loop_check); % initialize valid samples
valid_idx = 1;
error_samples = zeros(1,loop_check); % initialize error samples
error_idx = 1;
while valid_idx <= loop_check || error_idx <= loop_check
    temp = normrnd(1,sigma); % generate samples
    if temp > 0 % sort valid samples
        valid_samples(valid_idx) = temp;
        valid_idx = valid_idx + 1;
    else % sort error samples
        error_samples(error_idx) = temp;
        error_idx = error_idx + 1;
    end
end
```

# Sample Generation

- ▶ Samples are pulled from Gaussian distribution with a mean of 1 and a standard deviation of  $\sigma = \sqrt{\frac{1}{R \cdot 10^{SNR/10}}}$ , where  $R$  is the code rate
- ▶ Currently the tool generates a list of samples and sorts into valid and error sample bins

```
loop_check = get_error_sample_size(sym_size);  
valid_samples = zeros(1,loop_check); % initialize valid samples  
valid_idx = 1;  
error_samples = zeros(1,loop_check); % initialize error samples  
error_idx = 1;  
while valid_idx <= loop_check || error_idx <= loop_check  
    temp = normrnd(1,sigma); % generate samples  
    if temp > 0 % sort valid samples  
        valid_samples(valid_idx) = temp;  
        valid_idx = valid_idx + 1;  
    else % sort error samples  
        error_samples(error_idx) = temp;  
        error_idx = error_idx + 1;  
    end  
end
```



# Sample Generation

- ▶ Samples are pulled from Gaussian distribution with a mean of 1 and a standard deviation of  $\sigma = \sqrt{\frac{1}{R \cdot 10^{SNR/10}}}$ , where  $R$  is the code rate
- ▶ Currently the tool generates a list of samples and sorts into valid and error sample bins
- ▶ An error sample is one that comes from the negative tail of the Gaussian distribution

```
loop_check = get_error_sample_size(sym_size);
valid_samples = zeros(1,loop_check); % initialize valid samples
valid_idx = 1;
error_samples = zeros(1,loop_check); % initialize error samples
error_idx = 1;
while valid_idx <= loop_check || error_idx <= loop_check
    temp = normrnd(1,sigma); % generate samples
    if temp > 0 % sort valid samples
        valid_samples(valid_idx) = temp;
        valid_idx = valid_idx +1;
    else % sort error samples
        error_samples(error_idx) = temp;
        error_idx = error_idx +1;
    end
end
```

# Sample Generation

- ▶ Samples are pulled from Gaussian distribution with a mean of 1 and a standard deviation of  $\sigma = \sqrt{\frac{1}{R \cdot 10^{SNR/10}}}$ , where  $R$  is the code rate
- ▶ Currently the tool generates a list of samples and sorts into valid and error sample bins
- ▶ An error sample is one that comes from the negative tail of the Gaussian distribution
- ▶ There is probably a better way to do this, and finding that is on the to-do list

```
loop_check = get_error_sample_size(sym_size);
valid_samples = zeros(1,loop_check); % initialize valid samples
valid_idx = 1;
error_samples = zeros(1,loop_check); % initialize error samples
error_idx = 1;
while valid_idx <= loop_check || error_idx <= loop_check
    temp = normrnd(1,sigma); % generate samples
    if temp > 0 % sort valid samples
        valid_samples(valid_idx) = temp;
        valid_idx = valid_idx +1;
    else % sort error samples
        error_samples(error_idx) = temp;
        error_idx = error_idx +1;
    end
end
```

# Energy Calculation

- To create the model, all possible energy values must be calculated

```
%initialize Energy and check node matrices
E = zeros(2^sym_size,sym_size);
chk_nodes = ones(1, check_size);
chk_sum = zeros(1,sym_size);
% Calculate all possible energy values for each state
for row = 1:2^sym_size
    % Calculate all check nodes
    for adj_row = 1:check_size
        for adj_col = 1:sym_size
            if adj_mat(adj_row,adj_col) == 1
                chk_nodes(adj_row) = chk_nodes(adj_row)*x(row,adj_col);
                chk_sum(adj_col) = chk_sum(adj_col)+chk_nodes(adj_row);
            end
        end
    end
end
% Calculate energy values
for E_idx = 1:sym_size
```

# Energy Calculation

- ▶ To create the model, all possible energy values must be calculated
- ▶ The syndrome, called `chk_nodes` here, is also calculated

```
%initialize Energy and check node matrices
E = zeros(2^sym_size,sym_size);
chk_nodes = ones(1, check_size);
chk_sum = zeros(1,sym_size);
% Calculate all possible energy values for each state
for row = 1:2^sym_size
    % Calculate all check nodes
    for adj_row = 1:check_size
        for adj_col = 1:sym_size
            if adj_mat(adj_row,adj_col) == 1
                chk_nodes(adj_row) = chk_nodes(adj_row)*x(row,adj_col);
                chk_sum(adj_col) = chk_sum(adj_col)+chk_nodes(adj_row);
            end
        end
    end
end
% Calculate energy values
for E_idx = 1:sym_size
```

# Transition Probabilities

- Using the energy calculations, the transition matrix is generated

```
p = ones(2^sym_size,2^sym_size);  
% Flip probabilities calculated according to Eq 3.13 in T. Tithi  
% dissertation (pg. 26)  
for row = 1:2^sym_size  
    px = zeros(1,sym_size);  
    for p_idx = 1:sym_size  
        px(p_idx) = normcdf(theta,E(row,p_idx),sigma);  
    end  
    rowbin = dec2bin(row-1,sym_size);  
    for col = 1:2^sym_size  
        colbin = dec2bin(col-1,sym_size);  
        for p_idx = 1:sym_size  
            if rowbin(p_idx) == colbin(p_idx)  
                p(row,col) = p(row,col)*(1-px(p_idx));  
            else  
                p(row,col) = p(row,col)*px(p_idx);  
            end  
        end  
    end  
end
```

# Write Files and Process Outputs

- The model is written using the helper functions `write_model.m` and `write_explicit_model.m`

```
% Process Output for transient and steady state
if (tag(3) == 't' && tag(4) == 'r') || (tag(3) == 's' && tag(4)
    str_idx = regexp(output,regexptranslate('wildcard','0:\'\\(\\*\\
    output = substr(output,str_idx);
    split_output = strsplit(output,\'\\n\');
    for out_idx = 1:2^sym_size
        str_to_parse = char(split_output(out_idx));
        if (str_to_parse(1) >= "0") && (str_to_parse(1) <= "9")
            temp = textscan(str_to_parse,"%d:(%d)=%f");
```

# Write Files and Process Outputs

- ▶ The model is written using the helper functions `write_model.m` and `write_explicit_model.m`
- ▶ A system call to PRISM runs either a transient analysis, steady-state analysis, or a user-defined property analysis

```
% Process Output for transient and steady state
if (tag(3) == 't' && tag(4) == 'r') || (tag(3) == 's' && tag(4)
    str_idx = regexp(output,regexptranslate('wildcard','0:\'\\
    output = substr(output,str_idx);
    split_output = strsplit(output,\'\\n\');
    for out_idx = 1:2^sym_size
        str_to_parse = char(split_output(out_idx));
        if (str_to_parse(1) >= "0") && (str_to_parse(1) <= "9")
            temp = textscan(str_to_parse,"%d:(%d)=%f");
```

## Write Files and Process Outputs

- ▶ The model is written using the helper functions `write_model.m` and `write_explicit_model.m`
- ▶ A system call to PRISM runs either a transient analysis, steady-state analysis, or a user-defined property analysis
- ▶ The result is saved for transient and steady-state analysis

```
% Process Output for transient and steady state
if (tag(3) == 't' && tag(4) == 'r') || (tag(3) == 's' && tag(4)
    str_idx = regexp(output,regexptranslate('wildcard','0:\>(*\
    output = substr(output,str_idx);
    split_output = strsplit(output,"\n");
    for out_idx = 1:2^sym_size
        str_to_parse = char(split_output(out_idx));
        if (str_to_parse(1) >= "0") && (str_to_parse(1) <= "9")
            temp = textscan(str_to_parse,"%d:(%d)=%f");
```



# Write Files and Process Outputs

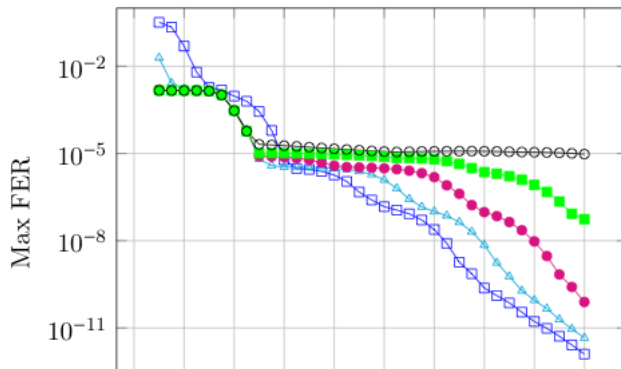
- ▶ The model is written using the helper functions `write_model.m` and `write_explicit_model.m`
- ▶ A system call to PRISM runs either a transient analysis, steady-state analysis, or a user-defined property analysis
- ▶ The result is saved for transient and steady-state analysis

▶ It works with some properties, but this needs to be expanded

```
% Process Output for transient and steady state
if (tag(3) == 't' && tag(4) == 'r') || (tag(3) == 's' && tag(4) == 's')
    str_idx = regexp(output,regexptranslate('wildcard','0:\>(*\n)'));
    output = substr(output,str_idx);
    split_output = strsplit(output,'\n');
    for out_idx = 1:length(split_output)
        str_to_parse = char(split_output(out_idx));
        if (str_to_parse(1) >= "0" && (str_to_parse(1) <= "9"))
            temp = textscan(str_to_parse,"%d:(%d)=%f");
```

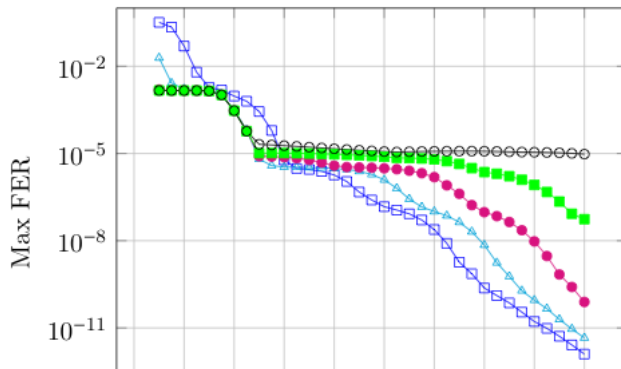
# What's Next

- This tool will hopefully automate all the manual calculations that were performed in T. Tithi's dissertation [1].



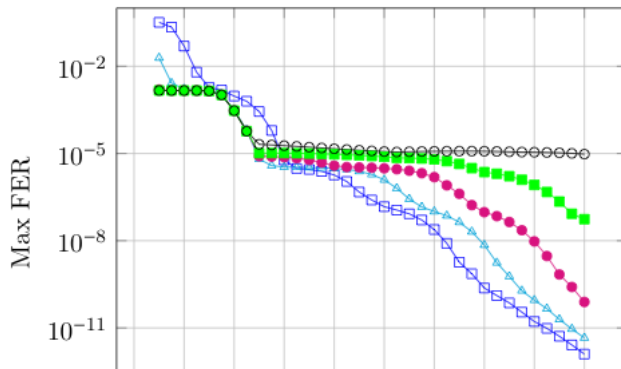
# What's Next

- ▶ This tool will hopefully automate all the manual calculations that were performed in T. Tithi's dissertation [1].
- ▶ The next step is to use the PRISM output to generate the FER graphs



# What's Next

- ▶ This tool will hopefully automate all the manual calculations that were performed in T. Tithi's dissertation [1].
- ▶ The next step is to use the PRISM output to generate the FER graphs
- ▶ Will use T. Tithi's analysis of the (8,8) absorbing set to verify validity



# Sources

- ▶ [1] T. Tithi, “Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding”, *All Graduate Theses and Dissertations*, pp. 7465, 2019.

# Sources

- ▶ [1] T. Tithi, “Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding”, *All Graduate Theses and Dissertations*, pp. 7465, 2019.
- ▶ [2] T. Tithi, C. Winstead, and G. Sundararajan, “Decoding LDPC codes via Noisy Gradient Descent Bit-Flipping with Re-Decoding”, 2015.

# Sources

- ▶ [1] T. Tithi, “Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding”, *All Graduate Theses and Dissertations*, pp. 7465, 2019.
- ▶ [2] T. Tithi, C. Winstead, and G. Sundararajan, “Decoding LDPC codes via Noisy Gradient Descent Bit-Flipping with Re-Decoding”, 2015.
- ▶ [3] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, “Gradient descent bit flipping algorithms for decoding LDPC codes”, *Communications, IEEE Transactions on*, vol. 58, no. 6, pp. 1610-1614, 2010.

# Sources

- ▶ [1] T. Tithi, “Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding”, *All Graduate Theses and Dissertations*, pp. 7465, 2019.
- ▶ [2] T. Tithi, C. Winstead, and G. Sundararajan, “Decoding LDPC codes via Noisy Gradient Descent Bit-Flipping with Re-Decoding”, 2015.
- ▶ [3] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, “Gradient descent bit flipping algorithms for decoding LDPC codes”, *Communications, IEEE Transactions on*, vol. 58, no. 6, pp. 1610-1614, 2010.
- ▶ [4] C. Winstead and E. Boutillon, “Hardware Demonstration of Noisy Gradient Descent Bit Flipping (NGDBF) for IEEE 802.3 Standard Code”.



# Sources

- ▶ [1] T. Tithi, “Error-Floors of the 802.3an LDPC Code for Noise Assisted Decoding”, *All Graduate Theses and Dissertations*, pp. 7465, 2019.
- ▶ [2] T. Tithi, C. Winstead, and G. Sundararajan, “Decoding LDPC codes via Noisy Gradient Descent Bit-Flipping with Re-Decoding”, 2015.
- ▶ [3] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, “Gradient descent bit flipping algorithms for decoding LDPC codes”, *Communications, IEEE Transactions on*, vol. 58, no. 6, pp. 1610-1614, 2010.
- ▶ [4] C. Winstead and E. Boutillon, “Hardware Demonstration of Noisy Gradient Descent Bit Flipping (NGDBF) for IEEE 802.3 Standard Code”.
- ▶ [5] “Trapping Set Ontology”,  
<https://uweb.engr.arizona.edu/~vasiclab/Projects/CodingTheory/TrappingSetOntology/>  
(accessed Aug 4, 2023).