Signal Processing with Machine Learning Techniques

Naman Gupta

Automatic Differentiation Variational Inference (ADVI) and STAN

Ideas from [this](#) paper, code [here](#)

STAN

- It is a probabilistic programming language with interfaces in R, Python etc.
- We particularly use STAN with Python interface.
- We make a STAN model in ‘.stan’ format and a python file(.py or .ipynb) through which we use and manipulate data.
- STAN model takes in the prior and likelihood distribution and minimises the loss function with respect to the parameters defined. In a way, returning the posterior distribution of these parameters.

Structure of STAN model

- Main blocks of a stan model are - functions, data, transformed data, parameters, transformed parameters, model, generated quantities (in this particular order).
- DATA - we give as input to the model
- PARAMETERS - these are what the model is fit on (or arguments wrt which the loss function is minimized)
- MODEL - this is where we define the priors and the likelihoods
- All others - transformed data, transformed parameters, generated quantities - are optional to specify.

SBL - Sparse Bayesian Learning

Ideas from this paper, code here

SBL - Sparse Bayesian Learning

- X is a sparse vector (M x 1), we are trying to infer and it follows $N(0, \alpha^{-1})$.
- Y is the observed data vector (N x 1), H is a NxM matrix and n is N-dimensional gaussian noise with variance as N_0, ie. $n \sim N(0, \sigma^2)$.
- We consider an underdetermined system with N=20 and M=50

$$\mathbf{Y = Hx + n}$$

- We also update alpha (the precision) iteratively as

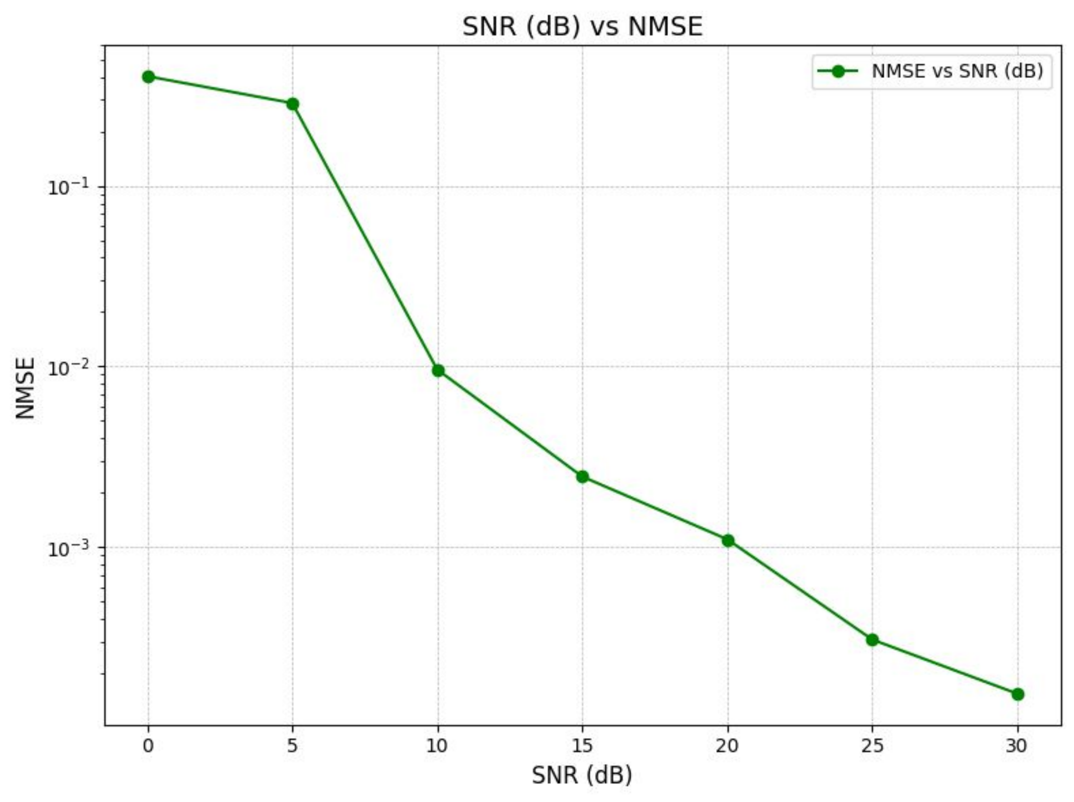
$$\boldsymbol{\alpha^{-1} = \mathrm{mean}(x)^2 + \mathrm{std}(x)}$$

- We vary N_0 (=)from 0dB to 30dB, to get the NMSE vs SNR plot.

STAN model

```
data {
  int<lower=1> N;         // Number of observations (rows of H)
  int<lower=1> M;         // Number of features (columns of H)
  vector[N] Y;            // Single observed data vector y
  matrix[N, M] H;         // Fixed measurement matrix
  real<lower=0> sigma;    // Noise standard deviation
}

parameters {
  vector[M] x;            // Sparse vector to infer
}

model {
  // Prior for x
  for (i in 1:M) {
    x[i] ~ normal(0, 1);
  }

  // Likelihood for Y: Gaussian noise with standard deviation sigma
  Y ~ normal(H * x, sigma);
}

```

Results



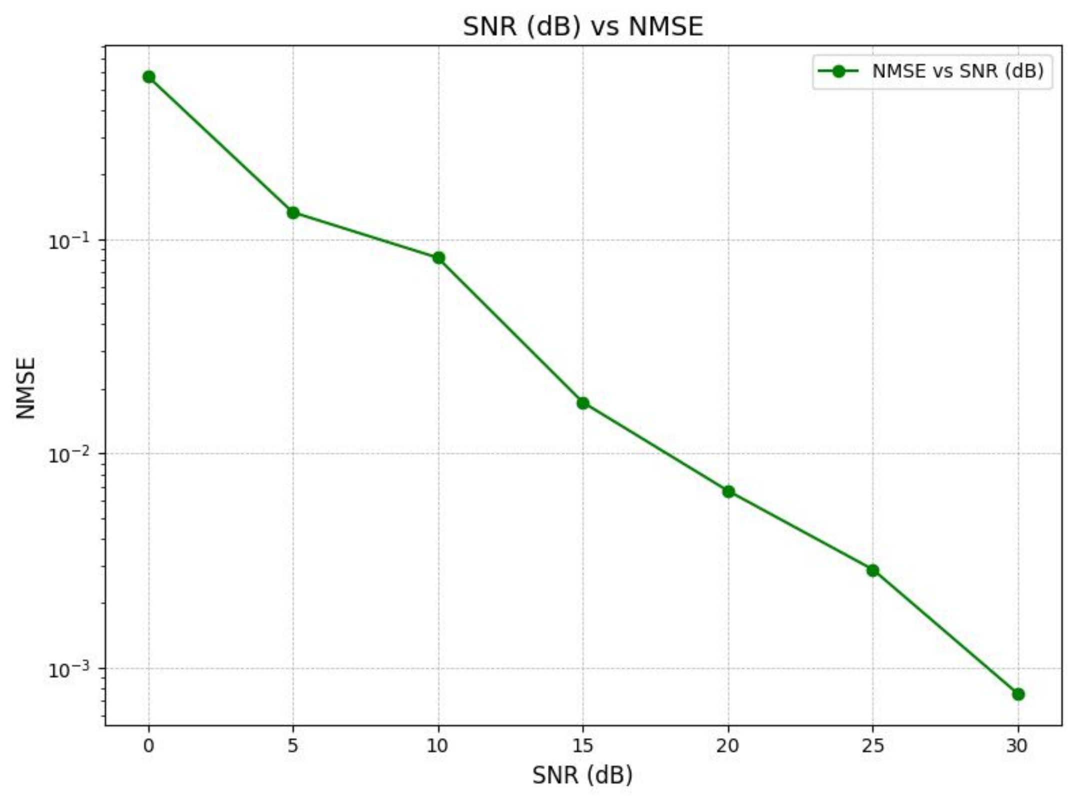

```
NMSE vs SNR
{0: 0.3277613593050331,
 5: 0.09983804555246331,
 10: 0.04339620147040254б,
 15: 0.006438327505046619,
 20: 0.003899392615703179б,
 25: 0.003374547729732350 4,
 30: 0.0003122690986882963б}
```

Using complex signals

```
data {
  int<lower=1> N;
  int<lower=1> M;
  vector[N] Y_real;
  vector[N] Y_imag;
  matrix[N, M] H_real;
  matrix[N, M] H_imag;
  real<lower=0> sigma;
  vector[M] alpha;
}

parameters {
  vector[M] x_real;
  vector[M] x_imag;
}

model {
  // Priors for x_real and x_imag
  for (i in 1:M) {
    x_real[i] ~ normal(0, alpha[i]);
    x_imag[i] ~ normal(0, alpha[i]);
  }

  // Likelihood for the real and imaginary parts of Y
  Y_real ~ normal(H_real * x_real - H_imag * x_imag, sigma);
  Y_imag ~ normal(H_real * x_imag + H_imag * x_real, sigma);
}
```

Defining ‘α’ as a latent variable

```
data {
  int<lower=1> N;
  int<lower=1> M;
  vector[N] Y;
  matrix[N, M] H;
  real<lower=0> sigma;
}

parameters {
  vector[M] x;
  vector<lower=1e-6>[M] alpha;
}

transformed parameters {
  vector[M] alpha_inv = sqrt(1/alpha);
}

model {
  // Prior for x
  for (i in 1:M) {
    x[i] ~ normal(0, alpha_inv[i]);
  }
  alpha ~ gamma(1e-6, 1e-6);
  // Likelihood for Y
  Y ~ normal(H * x, sigma);
}
```



Thoughts...

- We get comparatively higher NMSE, when ‘α’ is treated as a latent variable, but the time taken is longer (slightly but observable). It also reduces our efforts to update ‘α’ outside of stan.
- A smoother NMSE vs SNR is possible to get, but would have to average it over many runs.
- Generally, to get posterior once, with SNR=20dB, N=20, M=50, the model takes around 25-35 seconds to run.

Signal Classification (ADVI)

Ideas from this paper, code here

Signal Classification

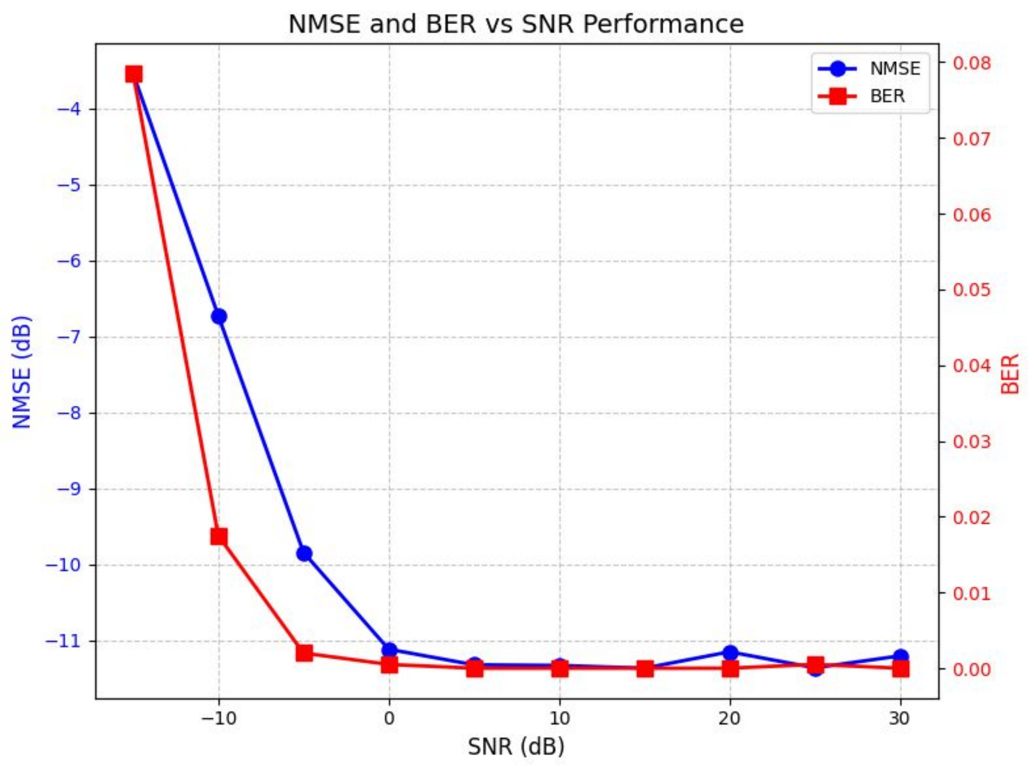
- $Y = \text{sign}(H^*x + n)$
- Y is Nx1 vector, H is NxM matrix, x is Mx1 vector, n is AWGN.
- ‘x’ has elements {+1, -1} and we get the estimated x from the model. We then calculate x_predicted = sign(x_estimated). We use ‘x’ and ‘x_predicted’ to calculate BER.
- Finally we get the BER vs SNR plot for SNR in range (-15dB to 30dB)

STAN model

```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
}
transformed parameters {
  vector[N] logits = H * x;
}
model {
  // Prior for x
  x ~ normal(0,1);

  // Likelihood
  Y ~ bernoulli_logit(logits);
}
```

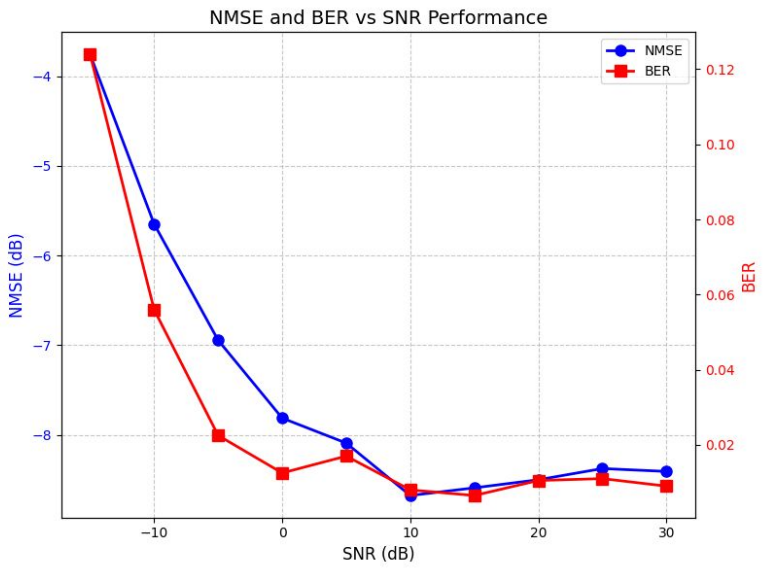
- Here, Y is a vector with {0,1} entries
- X is the inferred vector

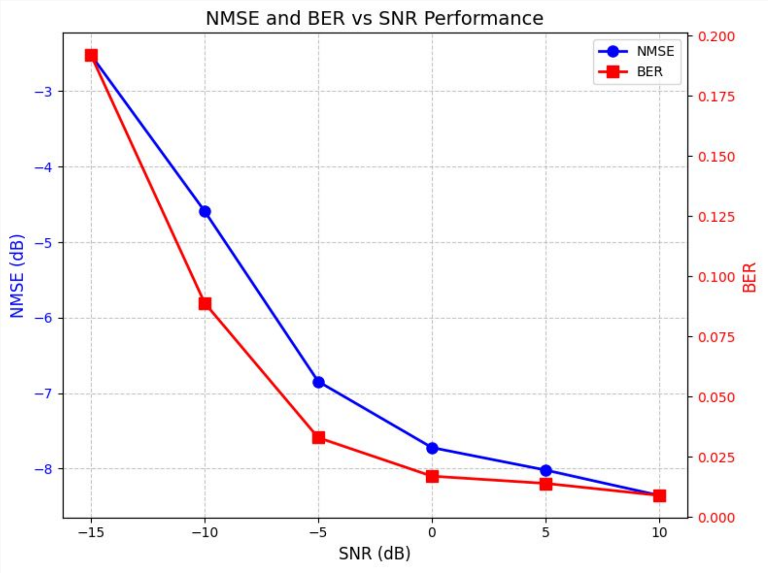
Results



- N = 128
- M = 20

Results



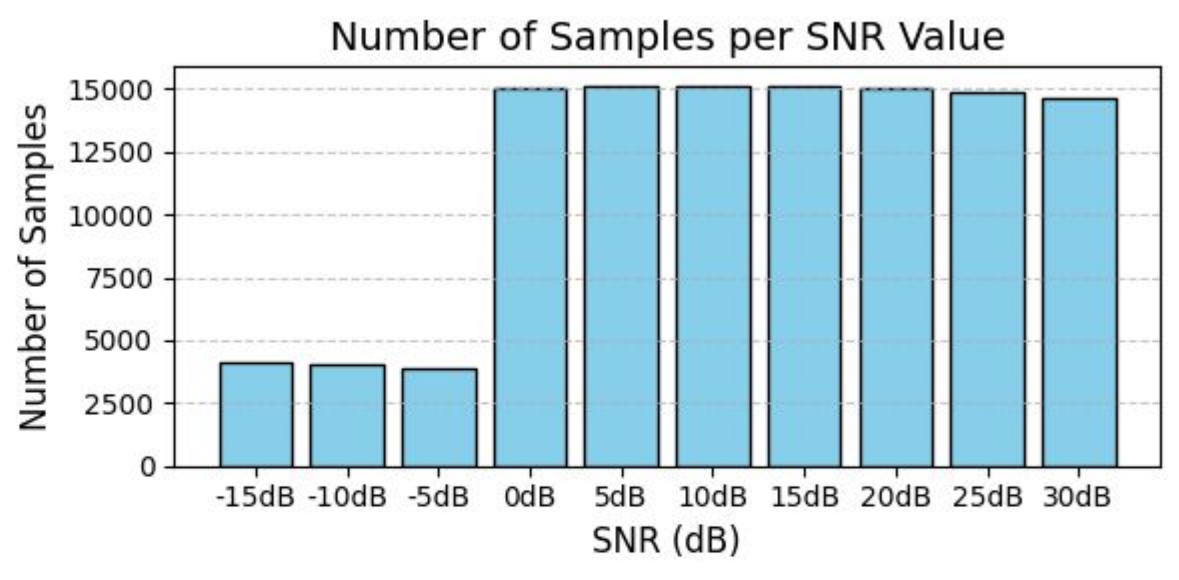


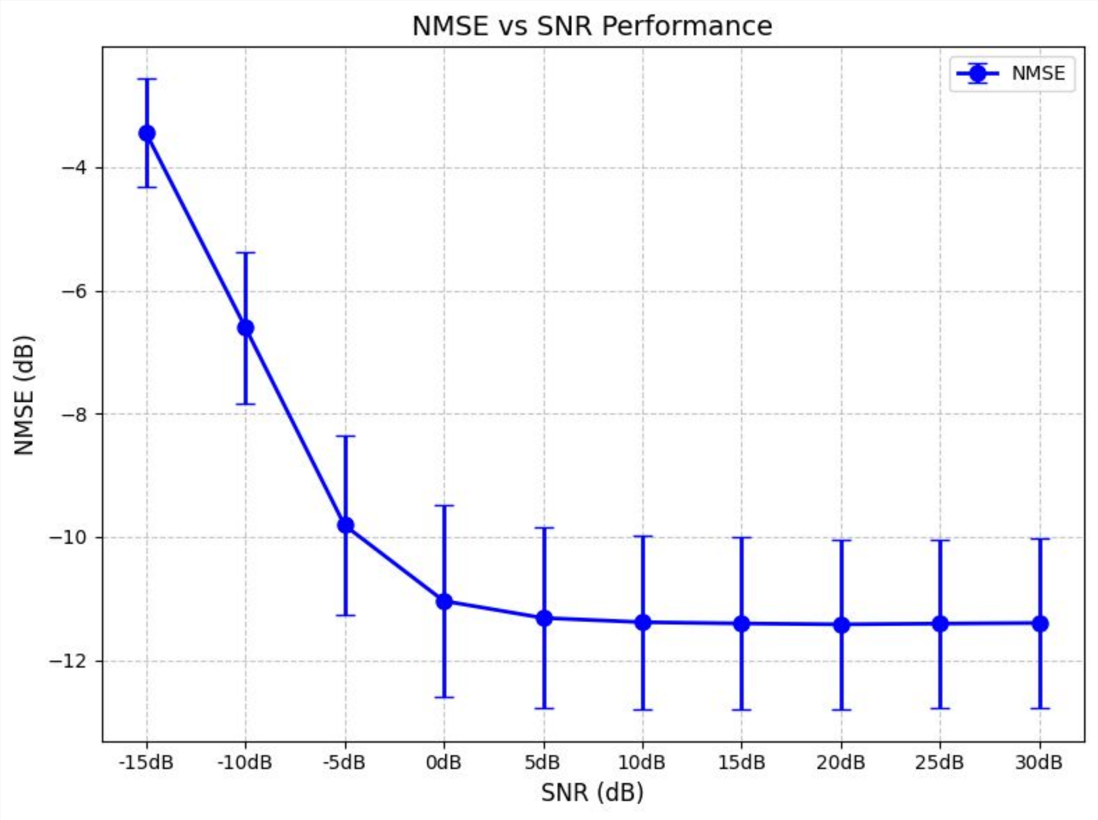
- N = 128
- M = 40

- N = 64
- M = 20

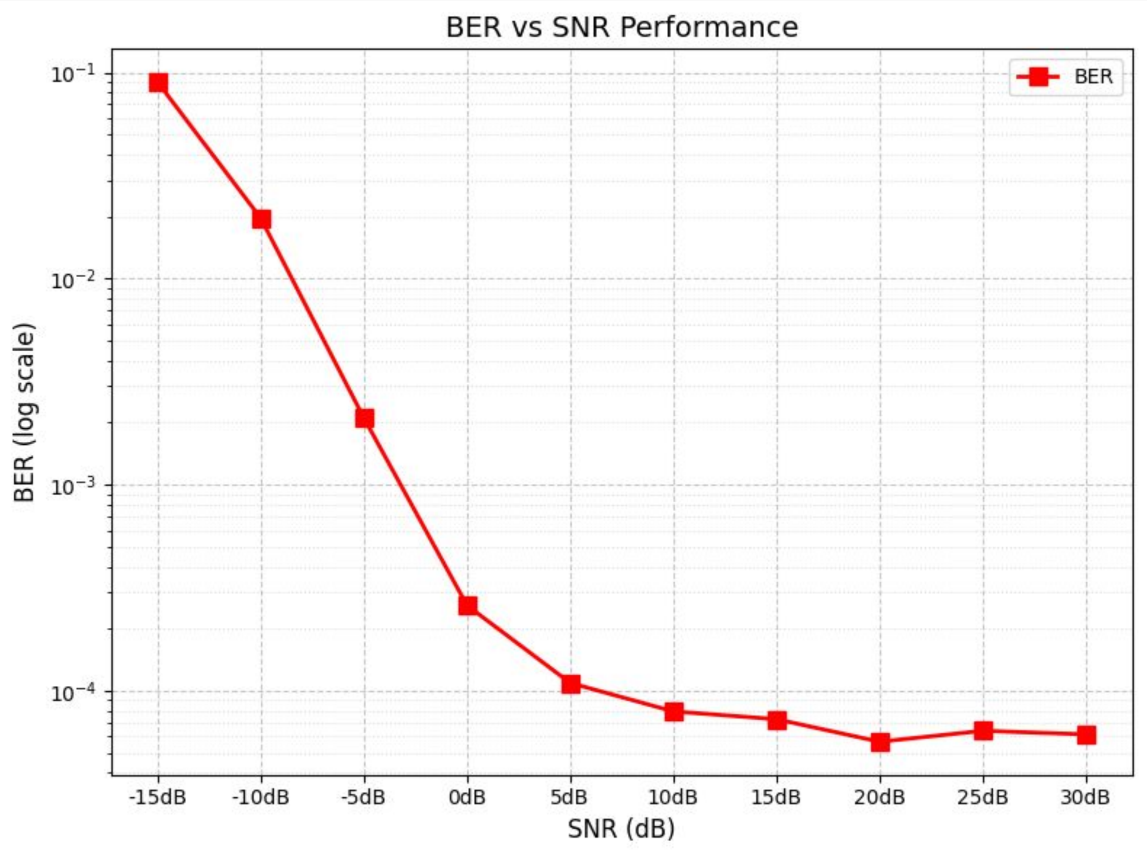
Final Results for

- N = 128, M = 20






```
Means =
{'-15dB': -3.4430280104152238,
  '-10dB': -6.5985468521759,
  '-5dB': -9.811372107823724,
  '0dB': -11.041803845578489,
  '5dB': -11.317221949022166,
  '10dB': -11.38660394540377,
  '15dB': -11.407116617037083,
  '20dB': -11.421109208533728,
  '25dB': -11.409321232620954,
  '30dB': -11.399125627029134}

StdDev =
{'-15dB': 0.8820421056641423,
  '-10dB': 1.2287833055532345,
  '-5dB': 1.4493985634288877,
  '0dB': 1.5577570781013421,
  '5dB': 1.4634484793213884,
  '10dB': 1.4093349267353794,
  '15dB': 1.3934962668780957,
  '20dB': 1.3797606684508361,
  '25dB': 1.3649404492048587,
  '30dB': 1.3771926574652007}
```



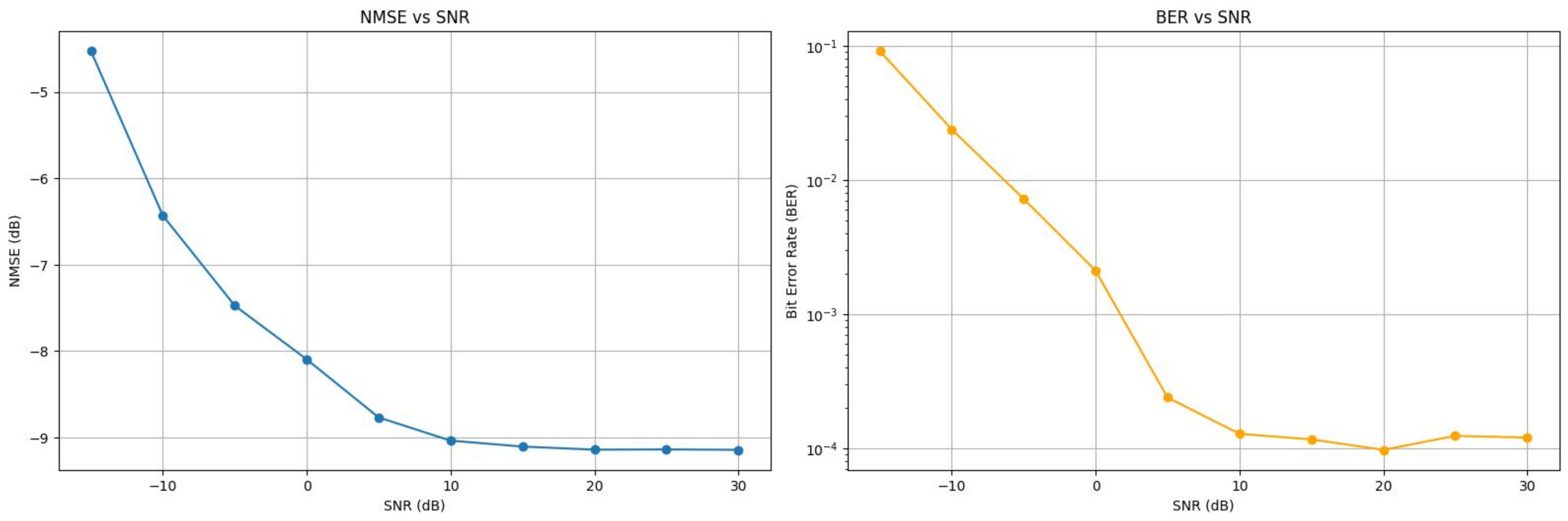

```
Means =
{'-15dB': 0.08994162004378496,
  '-10dB': 0.019672537831803522,
  '-5dB': 0.002109434927128611 6,
  '0dB': 0.00025972296217368143,
  '5dB': 0.00010915586133897858,
  '10dB': 7.938608097380259e-05,
  '15dB': 7.277057422598571e-05,
  '20dB': 5.656484993678047e-05,
  '25dB': 6.393431590281985e-05,
  '30dB': 6.144183506280722e-05}
```

Impulse Prior for Known Channel Data Detection

```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
}
transformed parameters {
  vector[N] logits = H * x;
}
model {
  for(m in 1:M){
    if(x[m] == 1) target += log(0.5);
    else if(x[m] == -1) target += log(0.5);
  }
  Y ~ bernoulli_logit(logits);
}
```

```
NMSE_Means =
{'-15dB': -4.528831,
  '-10dB': -6.433891,
  '-5dB': -7.475982,
  '0dB': -8.097237,
  '5dB': -8.769151,
  '10dB': -9.037724,
  '15dB': -9.106424,
  '20dB': -9.142239,
  '25dB': -9.139695,
  '30dB': -9.144127}
BER_Means =
{'-15dB': 0.091736,
  '-10dB': 0.023933,
  '-5dB': 0.007250,
  '0dB': 0.002115,
  '5dB': 0.000240,
  '10dB': 0.000129,
  '15dB': 0.000117,
  '20dB': 0.000098,
  '25dB': 0.000124,
  '30dB': 0.000121}
```

Impulse Prior for Known Channel Data Detection



Joint Channel Estimation and Data Detection using ADVI

Ideas from this paper, code here

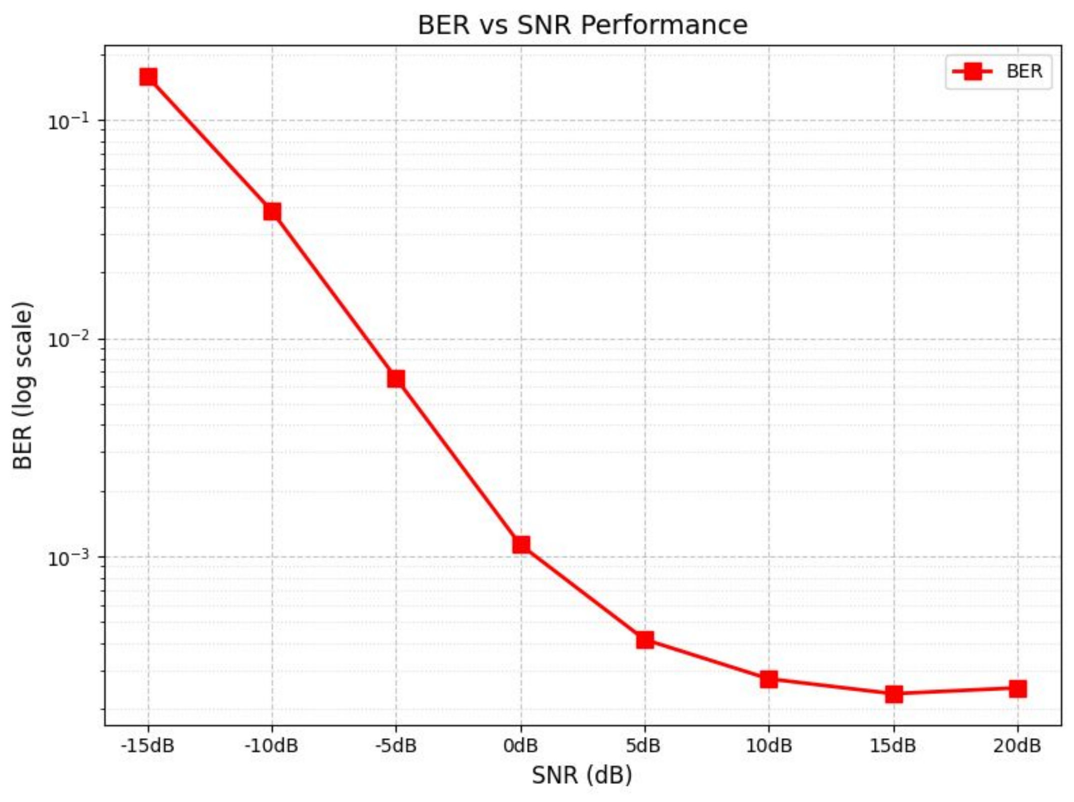
The Model

```
params = {
    'N': 128,
    'M': 20,
    'Tp': 80,
    'Td': 200,
    'Tc': 280,
}
```

$$\mathbf{Y = Hx + noise}$$

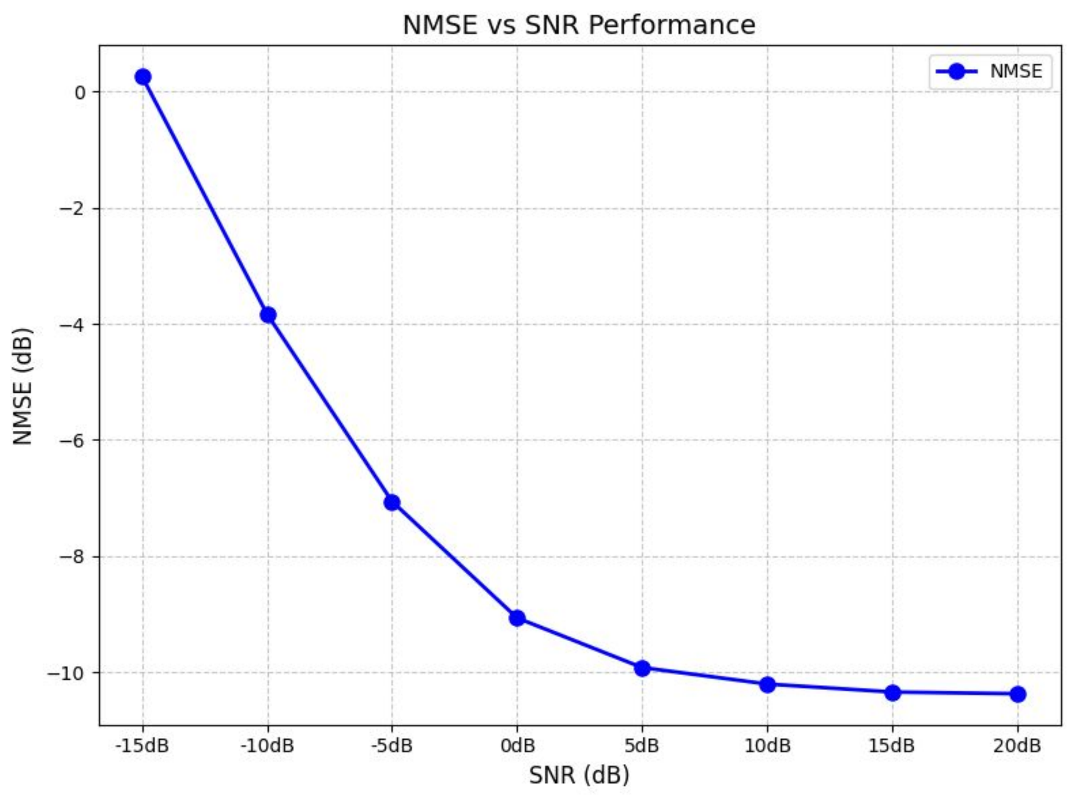
- Y is the received signal vector, dim(N, Tc)
- H is the channel matrix, dim(N, M)
- x is the transmitted signal, dim(M, Tc)
- Tp is the number of pilots (taken 80) and Td is the number of data vectors sent (taken 200).
- H ~ N(0,1) and noise is AWGN with N_0 noise power.

Results - BER for ADVI




```
Means =
{'-15dB': 0.1582333333333333,
  '-10dB': 0.0383333333333333,
  '-5dB': 0.0065666666666669,
  '0dB': 0.0011333333333333334,
  '5dB': 0.00041666666666666675,
  '10dB': 0.000275,
  '15dB': 0.000235,
  '20dB': 0.00025}
```

Results - NMSE




```
Means =
{'-15dB': 0.26011577153908405,
  '-10dB': -3.8310146378334955,
  '-5dB': -7.05703168694895,
  '0dB': -9.06263237920271,
  '5dB': -9.917335012980566,
  '10dB': -10.20116958762003l,
  '15dB': -10.338843542304598,
  '20dB': -10.368325146164285}
StdDev =
{'-15dB': 0.23306074492026693,
  '-10dB': 0.13007997459372095,
  '-5dB': 0.2289607807066905,
  '0dB': 0.12767117368850817,
  '5dB': 0.15711935012247571,
  '10dB': 0.18180210253522922,
  '15dB': 0.17551568495704106,
  '20dB': 0.17983971538919002}
```

SVM-based Channel Estimation and Data Detection

Ideas from this paper, code here

Proposed SVM based method

- For the system Y = Hx + noise, we have T_c entries of transmitted signal 'x', out of which T_p are known pilot sequences and rest T_d (= T_c - T_p) sequences are unknown.
- Y is the received signal (known) and H (channel) is unknown.
- Channel Estimation - Estimation H based on the T_p known sequences using soft-margin SVM
- Data Detection - Based on the estimated H, calculating the unknown part of 'x'.
- Re-estimating channel using the known 'x' and predicted 'x' together.
- Re-predicting 'x' and going on till H-estimation converges in a range.

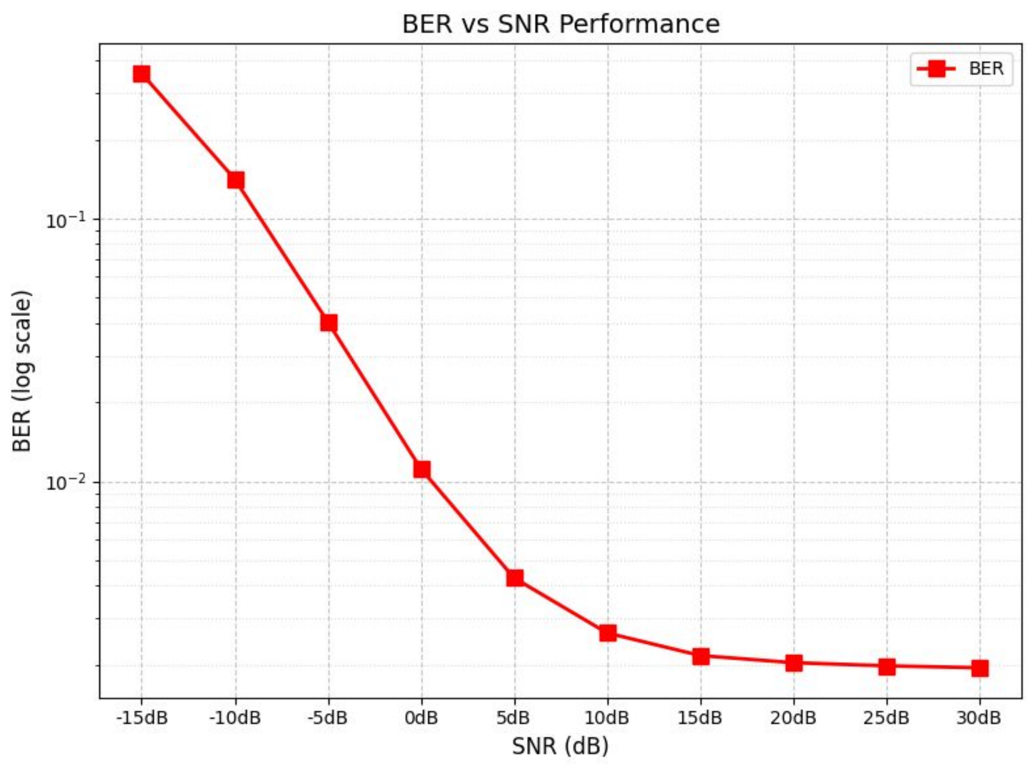
Model Specifications

```
params = {
    'N': 128,
    'M': 20,
    'Tp': 80,
    'Td': 120,
    'Tc': 200
}
```

$$\mathbf{Y = Hx + noise}$$

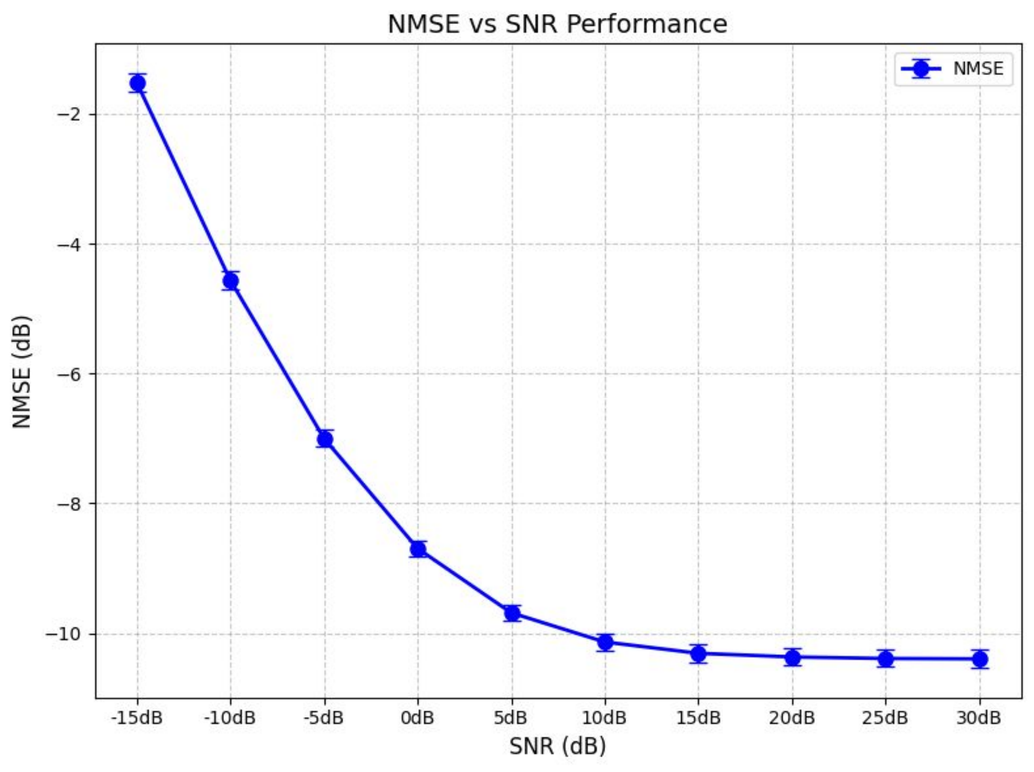
- Y is the received signal matrix, dim(N, T_c)
- H is the channel, dim(N, M)
- x is the transmitted signal, dim(M, T_c)
- Out of this, the pilot sequence(x_{known}) has dim(M, T_p) and the data sequence($x_{unknown}$) has dim(M, T_d)
- X is a random QAM symbol with unit power.
- H ~ N(0,1) and noise is AWGN with N_o noise power.

Results (SVM)




```
Means =
{'-15dB': 0.35761449480642116,
 '-10dB': 0.1416283050047214,
 '-5dB': 0.04049043909348439,
 '0dB': 0.011150849858356909,
 '5dB': 0.004284702549575039,
 '10dB': 0.002644003777148219,
 '15dB': 0.0021683191690273517,
 '20dB': 0.002036709159584482,
 '25dB': 0.001981507490636671,
 '30dB': 0.0019476481230212232}
```

Results (SVM)




```
Means =
{'-15dB': -1.515770569688407,
  '-10dB': -4.561528433201936,
  '-5dB': -6.996476278224057,
  '0dB': -8.69326520247198 8,
  '5dB': -9.684402120823497,
  '10dB': -10.134735316606227,
  '15dB': -10.308754128772248,
  '20dB': -10.365172485625326,
  '25dB': -10.388897990793923,
  '30dB': -10.394724807604804}
StdDev =
{'-15dB': 0.13873050295867018,
  '-10dB': 0.13225632543621907,
  '-5dB': 0.12634966379310883,
  '0dB': 0.1220252627517577,
  '5dB': 0.12728220032007947,
  '10dB': 0.13064808942603418,
  '15dB': 0.13232158194857907,
  '20dB': 0.1331332533331393,
  '25dB': 0.1305695784472933,
  '30dB': 0.1372341700010898 4}
```

Partitioned Variational Inference (PVI)

Ideas from [this](#) paper, code [here](#)

Partitioned Variational Inference

- It is a framework for probabilistic federated learning, where we assume there is a central server and several clients, all knowing a different set of data.
- So, every client, without sharing its data with other clients, updates the posterior distribution of the desired random variable on the server. Alongside, updating the likelihood of the data known to it.
- Such an algorithm ensures data privacy as well as a decentralized model for training.
- We have incorporated this idea, in a signal detection problem, where each client knows a subset of the entire channel, but all the clients transmit the same signal to the server.

Signal Detection with Known Channel using PVI

The Model

```
params = {
    'N': 128,
    'M': 20,
    'K': 16
}
```

$$\mathbf{Y = Hx + noise}$$

- Y is the received signal vector, dim(N, 1)
- H is the channel matrix, dim(N, M)
- x is the transmitted signal, dim(M, 1)
- K is the number of clients.
- Therefore each client knows exactly (N / K) rows of H matrix.
- H ~ N(0,1) and noise is AWGN with N_0 noise power.

The Algorithm

Algorithm 1 Partitioned Variational Inference

Input: data partition $\{\mathbf{y}_1, \dots, \mathbf{y}_M\}$, prior $p(\boldsymbol{\theta})$.

Initialise:

$$t_m^{(0)}(\boldsymbol{\theta}) := 1 \text{ for all } m = 1, 2, \dots, M.$$
$$q^{(0)}(\boldsymbol{\theta}) := p(\boldsymbol{\theta}).$$

for $i = 1, 2, \dots$ until convergence **do**

1 *Server: client selection step.*

$b_i :=$ set of indices of the next approximate likelihoods to refine.

Communicate $q^{(i-1)}(\boldsymbol{\theta})$ to each client in b_i.

2 *Client: update step.*

for $k \in b_i$ **do**

Compute the new local approximate posterior:

$$q_k^{(i)}(\boldsymbol{\theta}) := \underset{q(\boldsymbol{\theta}) \in \mathcal{Q}}{\arg\max} \int q(\boldsymbol{\theta}) \log \frac{q^{(i-1)}(\boldsymbol{\theta}) p(\mathbf{y}_k|\boldsymbol{\theta})}{q(\boldsymbol{\theta}) t_k^{(i-1)}(\boldsymbol{\theta})} \mathrm{d}\boldsymbol{\theta}.$$

Update the approximate likelihood:

$$t_k^{(i)}(\boldsymbol{\theta}) \propto \frac{q_k^{(i)}(\boldsymbol{\theta})}{q^{(i-1)}(\boldsymbol{\theta})} t_k^{(i-1)}(\boldsymbol{\theta})$$

Communicate $\Delta_k^{(i)}(\boldsymbol{\theta}) \propto \frac{t_k^{(i)}(\boldsymbol{\theta})}{t_k^{(i-1)}(\boldsymbol{\theta})}$ to server.

end for

3 *Server: update step.*

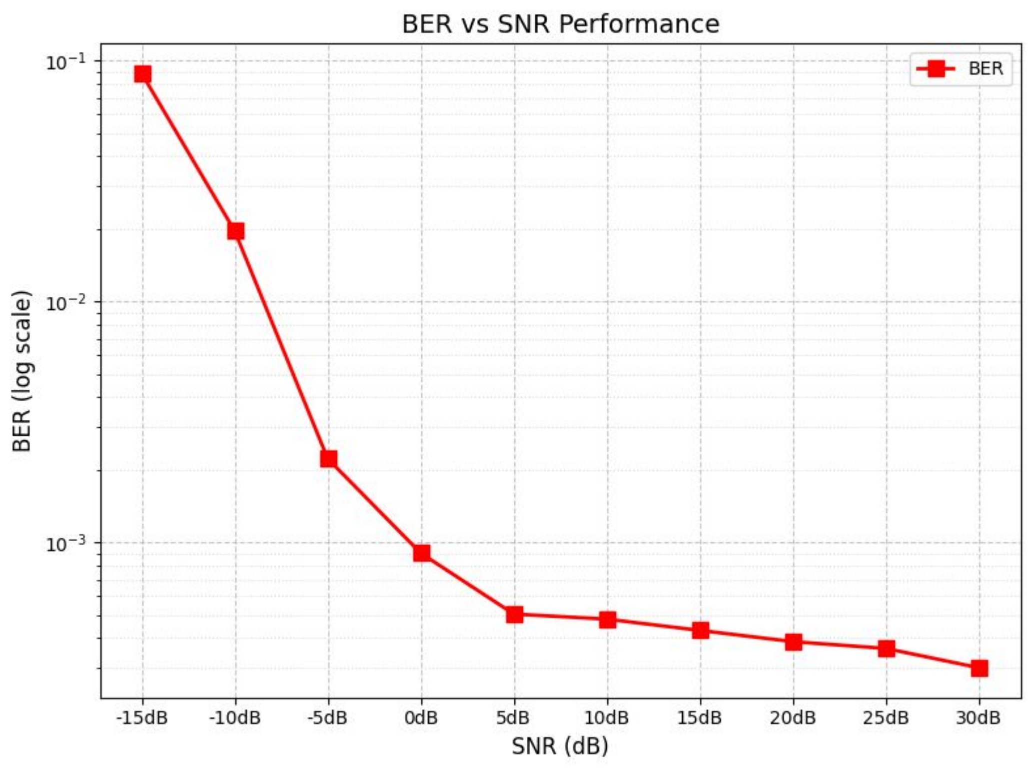
$$q^{(i)}(\boldsymbol{\theta}) \propto q^{(i-1)}(\boldsymbol{\theta}) \prod_{k \in b_i} \Delta_k^{(i)}(\boldsymbol{\theta}).$$

end for

3 Different Variations of the algorithm

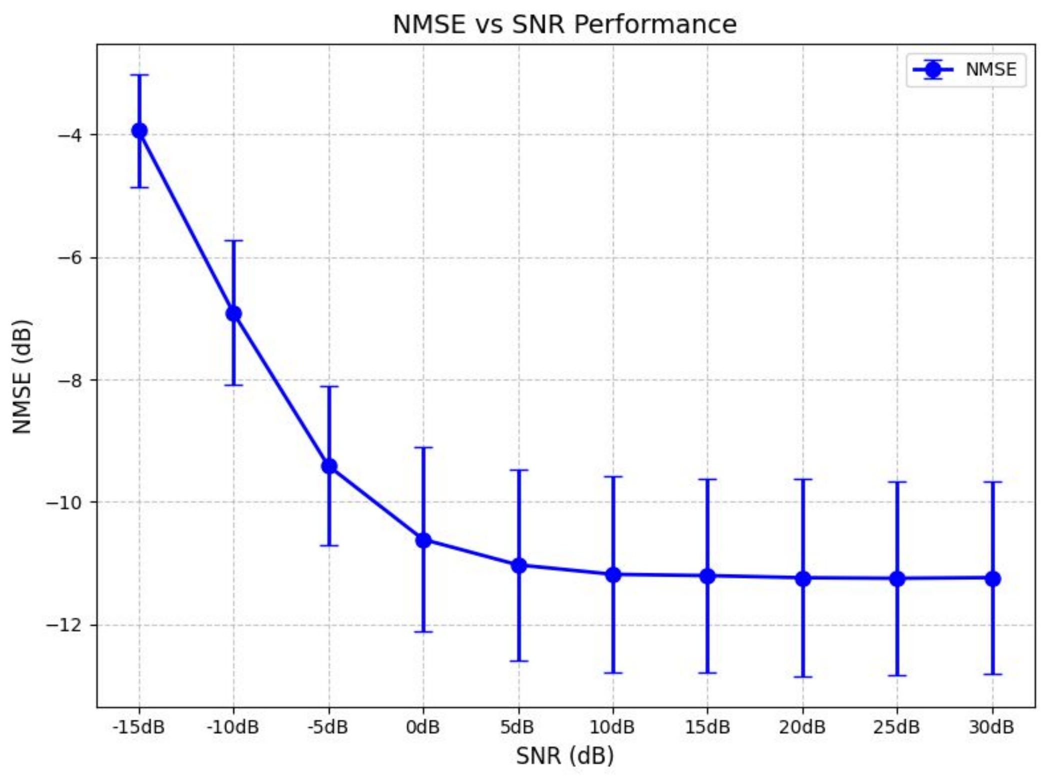
1. *sequential*, in which a single approximate likelihood is refined at each iteration;[1]
2. *synchronous*, in which all approximate likelihoods are refined at each iteration;
3. *asynchronous*, in which approximate likelihoods are updated whenever clients become available for performing local computation.

'Sequential' Model with Known Channel - BER



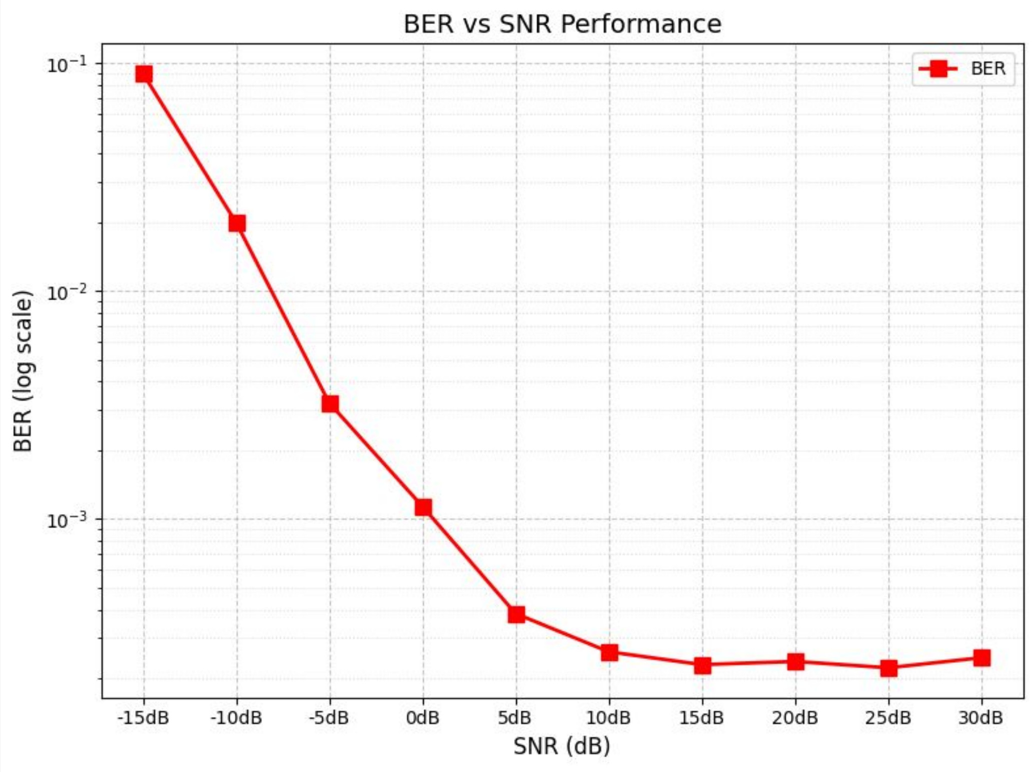

```
Means =
{'-15dB': 0.0883614088820827,
  '-10dB': 0.019598765432098764,
  '-5dB': 0.0022222222222222222,
  '0dB': 0.0009009009009009901,
  '5dB': 0.0005034101981162716,
  '10dB': 0.0004793510324483775,
  '15dB': 0.00043047783039173483,
  '20dB': 0.00038663352665009663,
  '25dB': 0.0003621807091117041,
  '30dB': 0.00030120481927710846}
```

‘Sequential’ Model with Known Channel - NMSE



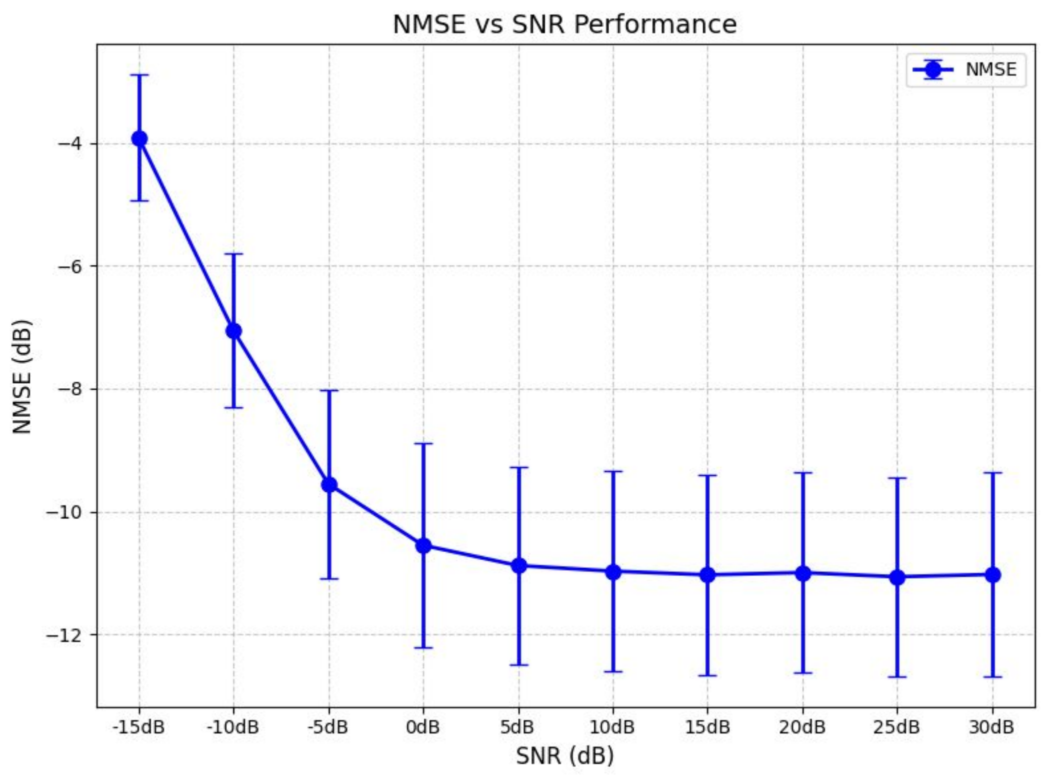

```
Means =
{'-15dB': -3.9330154636701327,
  '-10dB': -6.909514147073522,
  '-5dB': -9.409170126967398,
  '0dB': -10.61298632785521,
  '5dB': -11.029690805183101,
  '10dB': -11.184275865364535,
  '15dB': -11.205543393070913,
  '20dB': -11.239812170651943,
  '25dB': -11.249248053391492,
  '30dB': -11.238033958899992}
StdDev =
{'-15dB': 0.9151907372889666,
  '-10dB': 1.1838718121499512,
  '-5dB': 1.300727168098776,
  '0dB': 1.5103352234999914,
  '5dB': 1.5597341432886644,
  '10dB': 1.5966507921806132,
  '15dB': 1.586550323986087,
  '20dB': 1.612512146928695,
  '25dB': 1.5733424819544743,
  '30dB': 1.5611890820896661}
```

‘Synchronous’ Model with Known Channel - BER



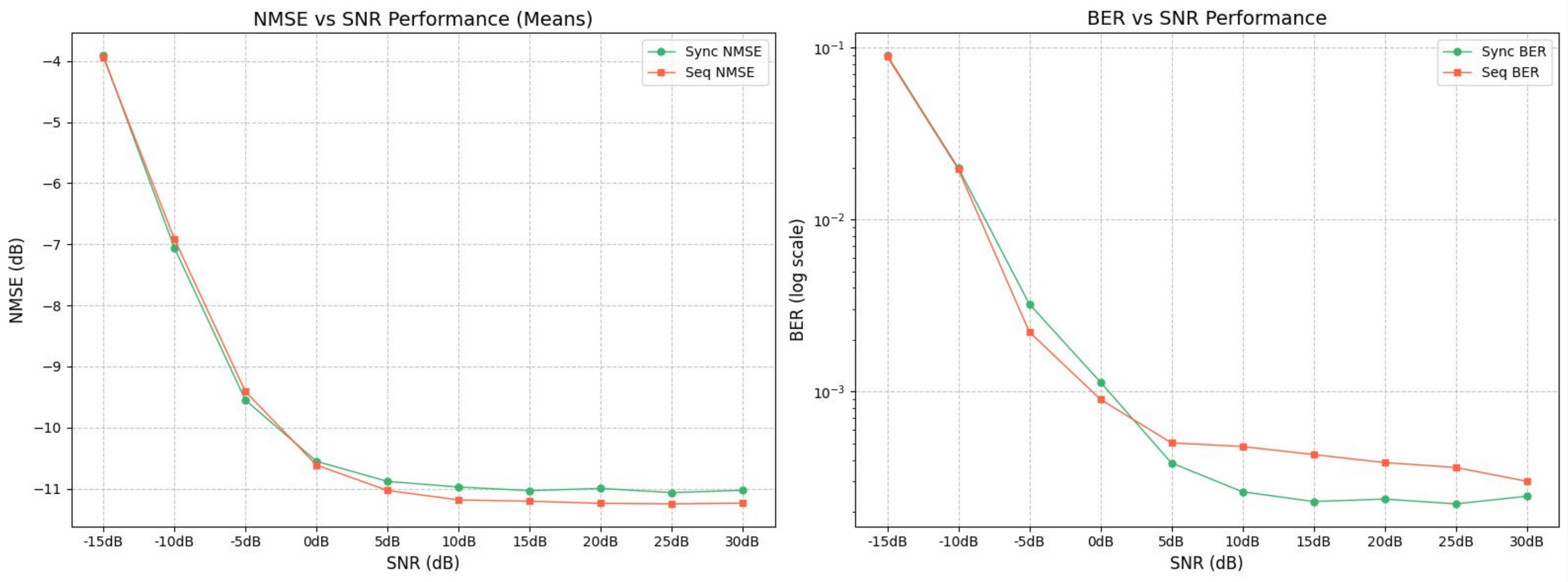

```
Means =
{'-15dB': 0.08984476067270375,
  '-10dB': 0.01994818652849741,
  '-5dB': 0.003225806451612904,
  '0dB': 0.00113555713271824,
  '5dB': 0.00038477982043608384,
  '10dB': 0.0002615233736515201,
  '15dB': 0.00022964509394572026,
  '20dB': 0.00023719165085388995,
  '25dB': 0.00022248590922574905,
  '30dB': 0.00024630541871921186}
```

'Synchronous' Model with Known Channel - NMSE




```
Means =
{'-15dB': -3.9090587528930123,
  '-10dB': -7.054317772176366,
  '-5dB': -9.548879254483387,
  '0dB': -10.551540145791318,
  '5dB': -10.881396874408676,
  '10dB': -10.973228634757918,
  '15dB': -11.032841413931754,
  '20dB': -10.996885749881335,
  '25dB': -11.065081998153262,
  '30dB': -11.026652140858792}
StdDev =
{'-15dB': 1.0257978755334818,
  '-10dB': 1.2495351561446515,
  '-5dB': 1.5349584817229862,
  '0dB': 1.6546339606734517,
  '5dB': 1.6138630859715575,
  '10dB': 1.635948171784257,
  '15dB': 1.6360548176802272,
  '20dB': 1.6291028097290186,
  '25dB': 1.61581238607458,
  '30dB': 1.6614563505065707}
```

Sequential vs **Synchronous**
Model with 'Known Channel'



Joint Channel Estimation and Data Detection using PVI

The Model

```
params = {
    'N': 128,
    'M': 20,
    'K': 32,
    'Tp': 80,
    'Td': 200,
    'Tc': 280,
}
```

$$\mathbf{Y = Hx + noise}$$

- Y is the received signal vector, dim(N, Tc)
- H is the channel matrix, dim(N, M)
- x is the transmitted signal, dim(M, Tc)
- Tp is the number of pilots (taken 80) and Td is the number of data vectors sent (taken 200).
- K is the number of clients.
- Therefore each client knows exactly (N / K) rows of H matrix.
- H ~ N(0,1) and noise is AWGN with N_0 noise power.

The Algorithm

Algorithm 1 Partitioned Variational Inference

Input: data partition $\{\mathbf{y}_1, \dots, \mathbf{y}_M\}$, prior $p(\boldsymbol{\theta})$.

Initialise:

$$t_m^{(0)}(\boldsymbol{\theta}) := 1 \text{ for all } m = 1, 2, \dots, M.$$
$$q^{(0)}(\boldsymbol{\theta}) := p(\boldsymbol{\theta}).$$

for $i = 1, 2, \dots$ until convergence **do**

1 *Server: client selection step.*

$b_i :=$ set of indices of the next approximate likelihoods to refine.

Communicate $q^{(i-1)}(\boldsymbol{\theta})$ to each client in b_i.

2 *Client: update step.*

for $k \in b_i$ **do**

Compute the new local approximate posterior:

$$q_k^{(i)}(\boldsymbol{\theta}) := \underset{q(\boldsymbol{\theta}) \in \mathcal{Q}}{\arg\max} \int q(\boldsymbol{\theta}) \log \frac{q^{(i-1)}(\boldsymbol{\theta}) p(\mathbf{y}_k|\boldsymbol{\theta})}{q(\boldsymbol{\theta}) t_k^{(i-1)}(\boldsymbol{\theta})} \mathrm{d}\boldsymbol{\theta}.$$

Update the approximate likelihood:

$$t_k^{(i)}(\boldsymbol{\theta}) \propto \frac{q_k^{(i)}(\boldsymbol{\theta})}{q^{(i-1)}(\boldsymbol{\theta})} t_k^{(i-1)}(\boldsymbol{\theta})$$

Communicate $\Delta_k^{(i)}(\boldsymbol{\theta}) \propto \frac{t_k^{(i)}(\boldsymbol{\theta})}{t_k^{(i-1)}(\boldsymbol{\theta})}$ to server.

end for

3 *Server: update step.*

$$q^{(i)}(\boldsymbol{\theta}) \propto q^{(i-1)}(\boldsymbol{\theta}) \prod_{k \in b_i} \Delta_k^{(i)}(\boldsymbol{\theta}).$$

end for

3 Different Variations of the algorithm

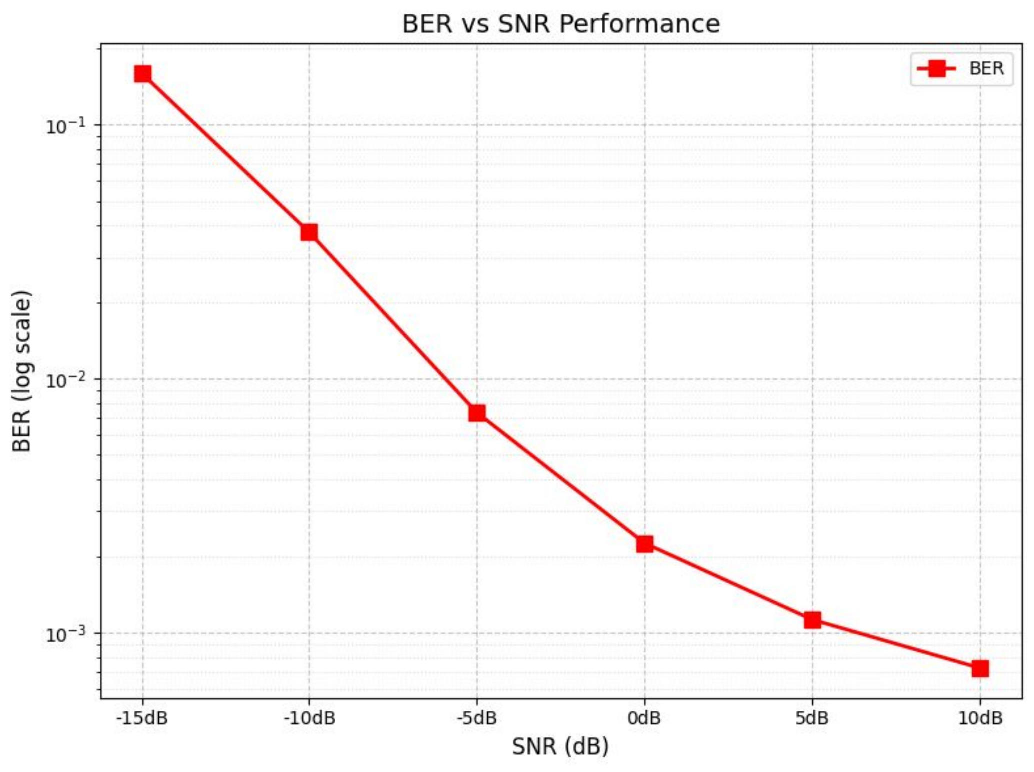
1. *sequential*, in which a single approximate likelihood is refined at each iteration;[1]
2. *synchronous*, in which all approximate likelihoods are refined at each iteration;
3. *asynchronous*, in which approximate likelihoods are updated whenever clients become available for performing local computation.

The Parameters

```
training_params_H = {
    'lr': 3e-2,
    'batch': 10000,
    'epochs': 20,
    'loops': 50,
    'lambda': 0.95,
    'stop': 5e-4
}

training_params_x = {
    'lr': 3e-2,
    'batch': 10000,
    'epochs': 10,
    'loops': 30,
    'lambda': 0.95,
    'stop': 1e-3
}
```

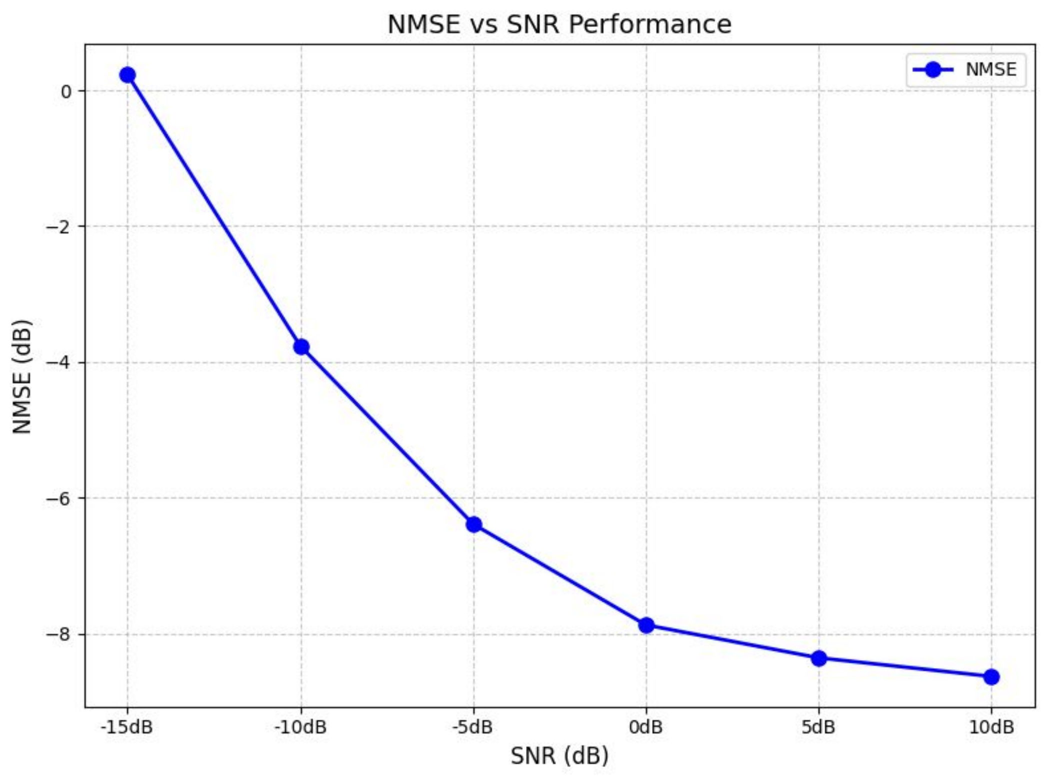
```
params = {
     'N' = 128
     'M' = 20
     'K' = 32
     'Tp' = 80
     'Td' = 200
     'Tc' = 280
}
```

‘Sequential’ Model for CEDD - BER



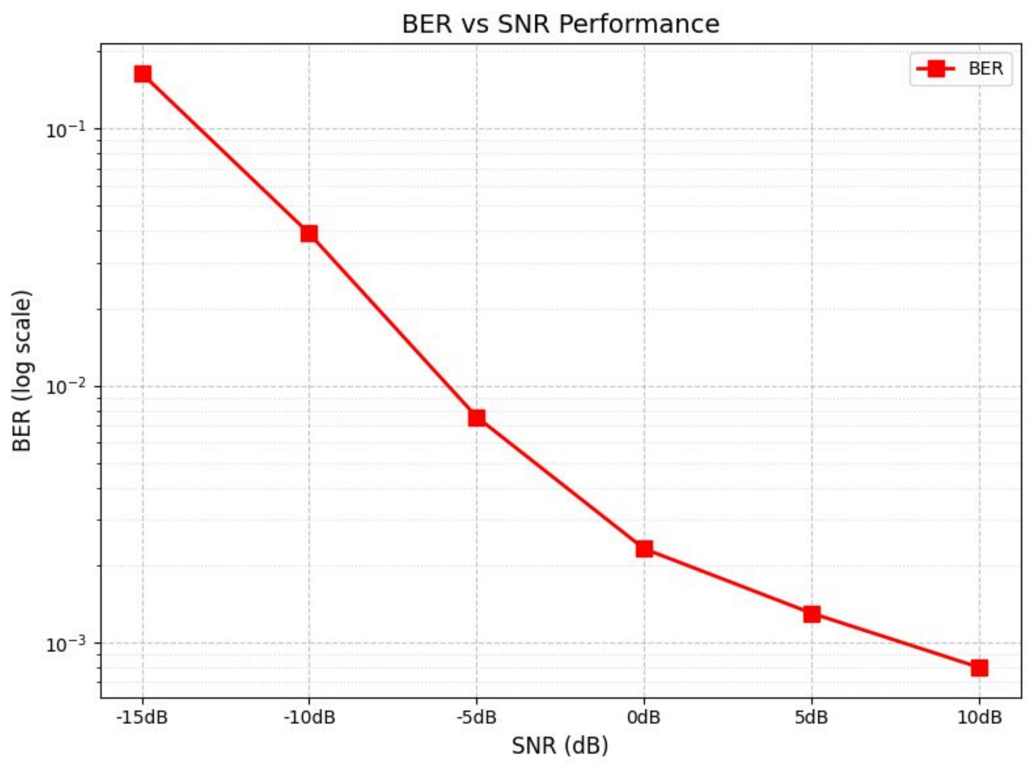

```
Means =
{'-15dB': 0.15944999999999998,
  '-10dB': 0.0377,
  '-5dB': 0.007350000000000001,
  '0dB': 0.00225,
  '5dB': 0.0011250000000000001,
  '10dB': 0.0007250000000000002}
```

'Sequential' Model for CEDD - NMSE



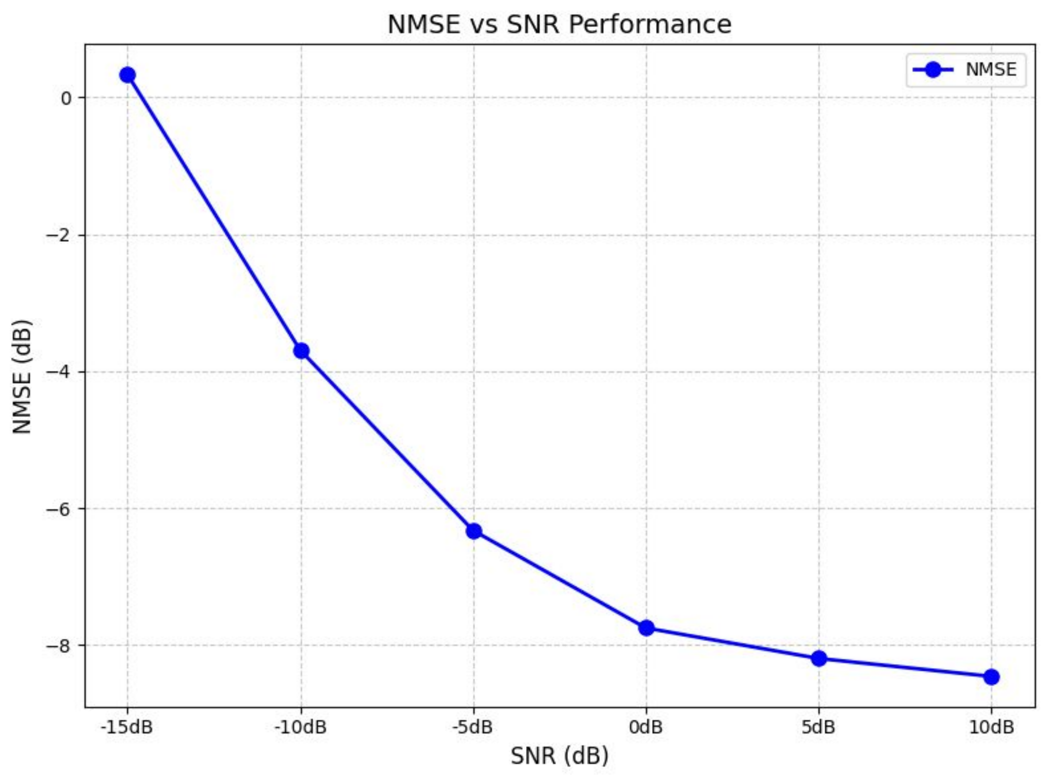

```
Means =
{'-15dB': 0.23605194642897814,
  '-10dB': -3.762195365144591,
  '-5dB': -6.3868753444314175,
  '0dB': -7.873957952862594,
  '5dB': -8.358459293790983,
  '10dB': -8.632693611206019}
StdDev =
{'-15dB': 0.08661759826681711,
  '-10dB': 0.22283365032437882,
  '-5dB': 0.23714779143198442,
  '0dB': 0.13961798620190574,
  '5dB': 0.17882661126052413,
  '10dB': 0.16177412314135228}
```

'Synchronous' Model for CEDD - BER



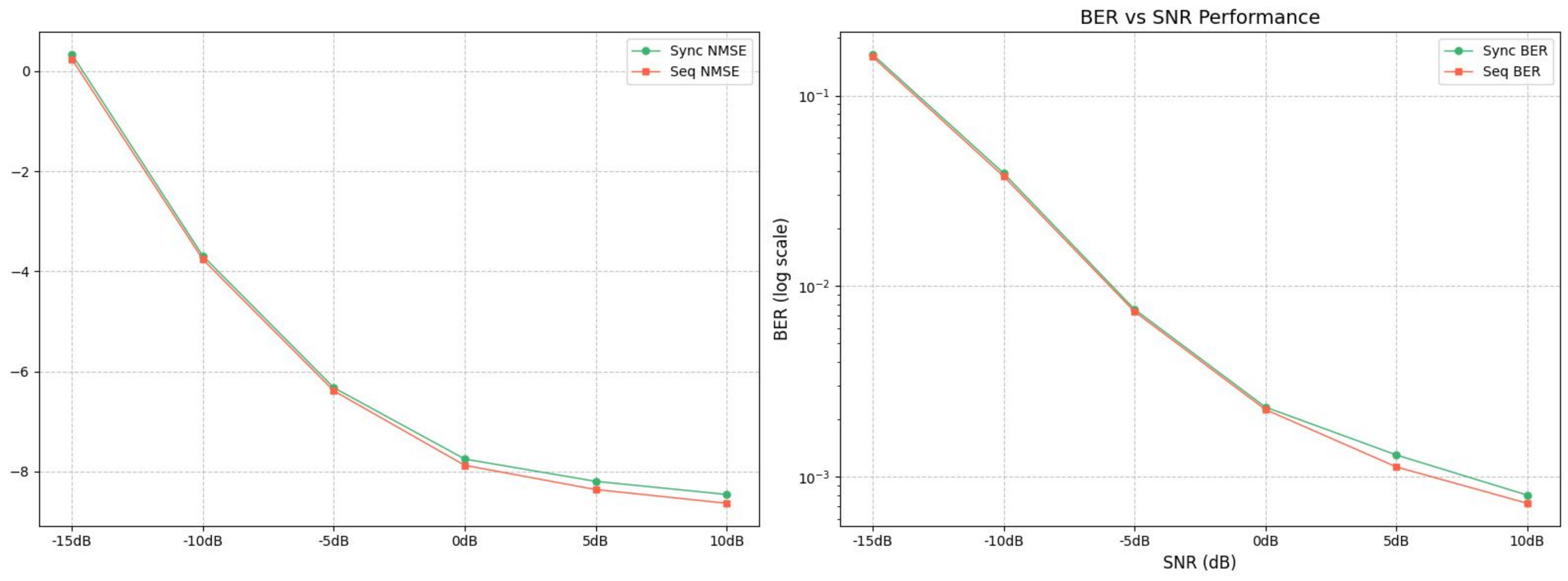

```
Means =
{'-15dB': 0.16410000000000002,
  '-10dB': 0.0391,
  '-5dB': 0.007550000000000001,
  '0dB': 0.0023166666666666665,
  '5dB': 0.0013000000000000002,
  '10dB': 0.0008}
```

‘Synchronous’ Model for CEDD - NMSE




```
Means =
{'-15dB': 0.3342038365781332,
  '-10dB': -3.6919397522054624,
  '-5dB': -6.323976055897844,
  '0dB': -7.748905739133786,
  '5dB': -8.194109277150604,
  '10dB': -8.456403164281657}
StdDev =
{'-15dB': 0.20341772487867268,
  '-10dB': 0.21638030290185814,
  '-5dB': 0.2544558678876974,
  '0dB': 0.15718779859367593,
  '5dB': 0.1744179087433486,
  '10dB': 0.1473413944615679}
```

Sequential vs **Synchronous**
Model for ‘Joint CE and DD’



Different Priors for Sparse Signal

Some Sparsity Inducing Priors

1) **Spike and Slab Prior**

   The spike and slab prior is considered the "gold standard" for Bayesian variable selection. It uses a mixture of a point mass at zero (the spike) and a continuous distribution (the slab) for non-zero coefficients.

   Research papers:

   - "Bayesian Variable Selection Using Spike and Slab Priors" by Ishwaran and Rao (2005)

2) **Laplace Prior**

   The Laplace prior, also known as the double exponential prior, is equivalent to L1 regularization (Lasso) in a Bayesian framework.

   Research paper:

   - "Bayesian Lasso Regression" by Park and Casella (2008)

3) **Structured Sparsity Priors**

   These priors incorporate more complex structural information about the sparsity patterns.

   Research paper:

   - "Structured Variable Selection with Sparsity-Inducing Norms" by Jenatton et al. (2011)

4) **Dirichlet-Laplace Prior**

   This prior combines the Dirichlet distribution with the Laplace distribution to induce sparsity.

   Research paper:

   - "The Bayesian Bridge" by Polson et al. (2014)

(continued...)

5) **<u>Horseshoe Prior</u>**

The horseshoe prior is a continuous shrinkage prior that allows strong shrinkage for noise variables while leaving large signals unshrunk.

Research papers:

- "Handling Sparsity via the Horseshoe" by Carvalho et al. (2009)
- "The Horseshoe + Estimator of Ultra-Sparse Signals" by Bhadra et al. (2017)

6) **<u>Sparsity-Inducing Categorical Prior</u>**

This prior is specifically designed for improving robustness in classification tasks using the Information Bottleneck framework.

Research papers:

- "Sparsity-Inducing Categorical Prior Improves Robustness of the Information Bottleneck" by Samaddar et al. (2022)

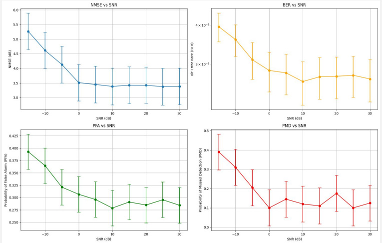
7) **<u>Group Sparsity Priors</u>**

These priors encourage sparsity at the group level, useful when variables have a natural grouping structure.

Research papers:

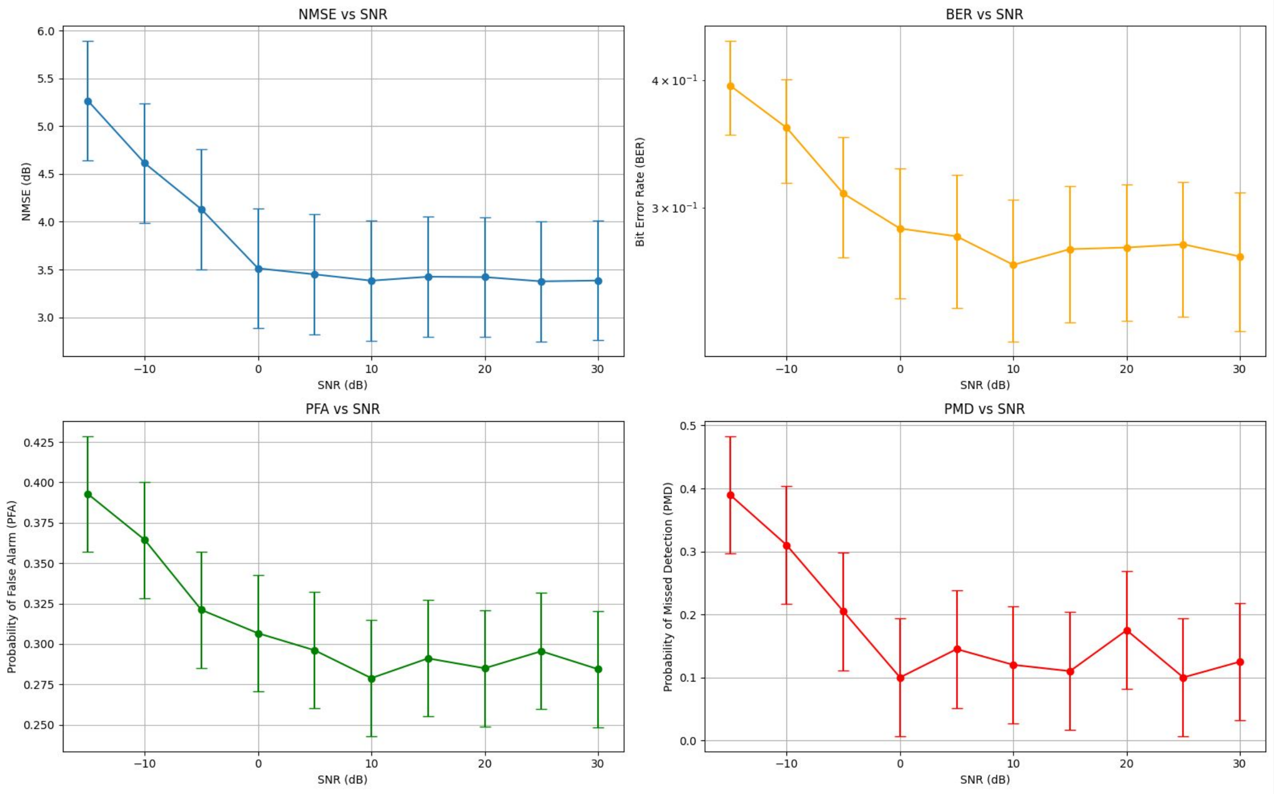
- "The Group Lasso for Logistic Regression" by Meier et al. (2008)
- "Bayesian Group Lasso for Gene Expression Networks" by Xu et al. (2015)

#1 Gaussian Prior




```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
}
transformed parameters {
  vector[N] logits = H * x;
}
model {
  x ~ normal(0, 1);
  Y ~ bernoulli_logit(logits);
}
```

#1 Gaussian Prior

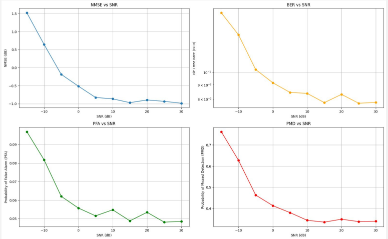



```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
}
transformed parameters {
  vector[N] logits = H * x;
}
model {
  x ~ normal(0, 1);
  Y ~ bernoulli_logit(logits);
}
```

#2 Gaussian with latent ‘α’

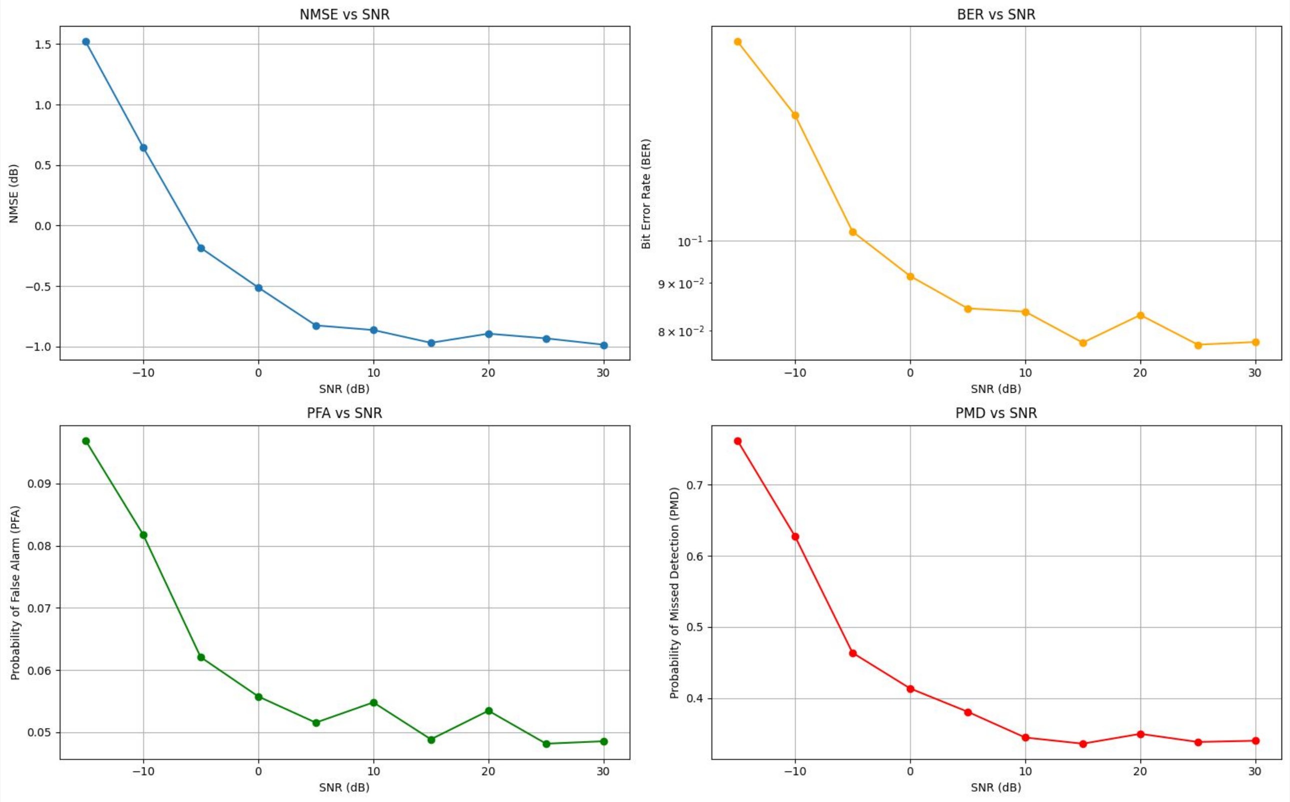
```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
  vector<lower=1e-6>[M] alpha;
}
transformed parameters {
  vector[M] alpha_inv = sqrt(1/alpha);
  vector[N] logits = H * x;
}
model {
  x ~ normal(0, alpha_inv);
  alpha ~ gamma(1e-6, 1e-6);
  Y ~ bernoulli_logit(logits);
}
```

#3 Impulse Prior



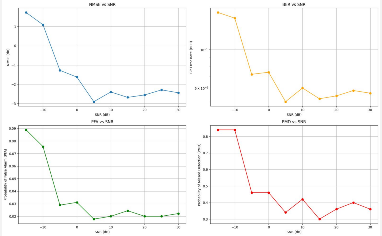

```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
  vector<lower=0, upper=1>[M] theta1, theta2, thetaa3;
}
transformed parameters {
  vector<lower=0, upper=1>[M] theta1_, theta2_, theta3_;
  for(m in 1:M){
    theta1_[m] = theta1[m] / (theta1[m] + theta2[m] + theta3[m]);
    theta2_[m] = theta2[m] / (theta1[m] + theta2[m] + theta3[m]);
    theta3_[m] = theta3[m] / (theta1[m] + theta2[m] + theta3[m]);
  }
  vector[N] logits = H * x;
}
model {
  theta1_ ~ beta(2,5); theta2_ ~ beta(2,5); theta3_ ~ beta(5,2);
  for (m in 1:M) {
    target += log(theta3_[m]) + normal_lpdf(x[m] | 0, 1);
    target += log(theta2_[m]) + normal_lpdf(x[m] | -1, 1);
    target += log(theta1_[m]) + normal_lpdf(x[m] | 1, 1);
  }
  Y ~ bernoulli_logit(logits);
}
```

#3 Impulse Prior



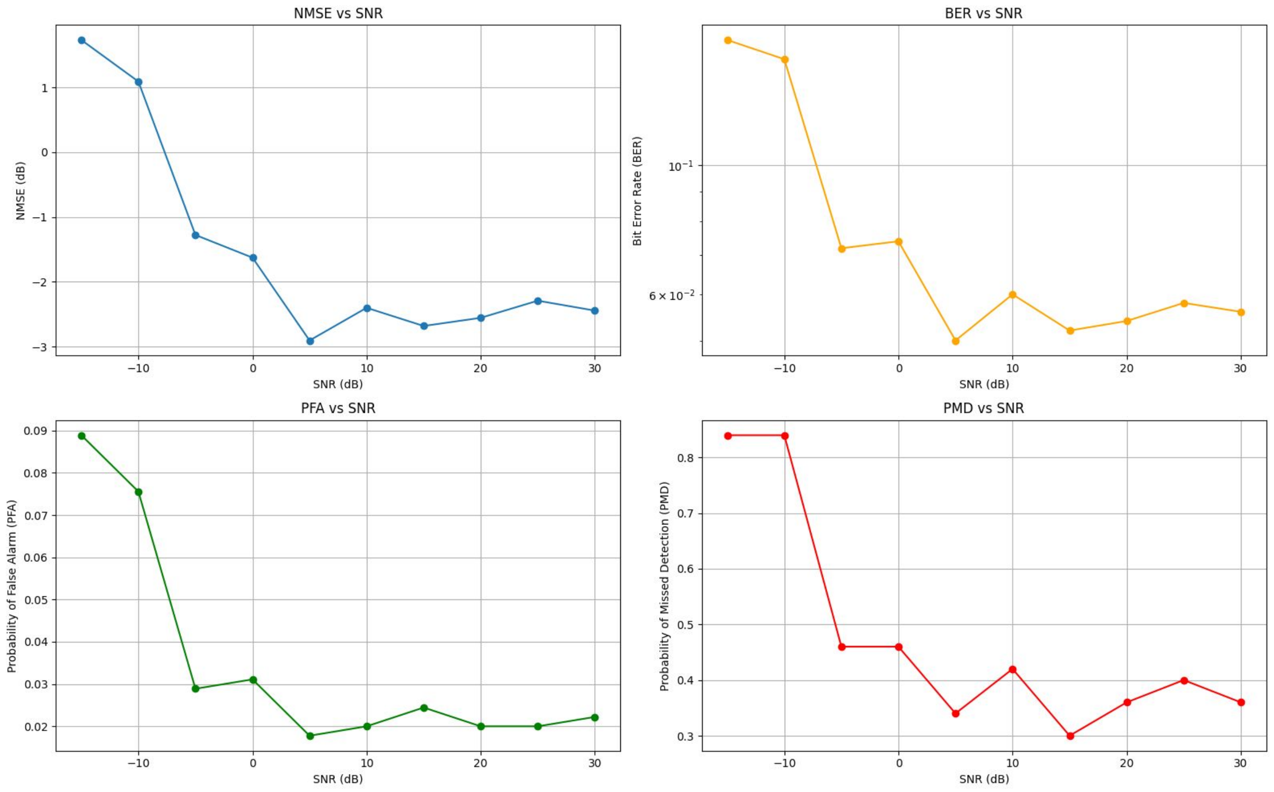

```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
  vector<lower=0, upper=1>[M] theta1, theta2, thetaa3;
}
transformed parameters {
  vector<lower=0, upper=1>[M] theta1_, theta2_, theta3_;
  for(m in 1:M){
    theta1_[m] = theta1[m] / (theta1[m] + theta2[m] +
theta3[m]);
    theta2_[m] = theta2[m] / (theta1[m] + theta2[m] +
theta3[m]);
    theta3_[m] = theta3[m] / (theta1[m] + theta2[m] +
theta3[m]);
  }
  vector[N] logits = H * x;
}
model {
  theta1_ ~ beta(2,5); theta2_ ~ beta(2,5); theta3_ ~
beta(5,2);
  for (m in 1:M) {
    target += log(theta3_[m]) + normal_lpdf(x[m] | 0, 1);
    target += log(theta2_[m]) + normal_lpdf(x[m] | -1,
1);
    target += log(theta1_[m]) + normal_lpdf(x[m] | 1, 1);
  }
  Y ~ bernoulli_logit(logits);
}
```

#4 Spike and Slab Prior




```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
  vector<lower=0, upper=1>[M] theta;
  vector<lower=1e-6>[M] alpha;
}
transformed parameters {
  vector[M] alpha_inv = sqrt(1/alpha);
  vector[N] logits = H * x;
}
model {
  alpha ~ gamma(1e-3, 1e-3);
  for (m in 1:M) {
    target += log(1 - theta[m]) + normal_lpdf(x[m] | 0, alpha_inv[m]);
    target += log(theta[m]) + normal_lpdf(x[m] | 0, 1);
  }
  theta ~ beta(1,1);
  Y ~ bernoulli_logit(logits);
}
```

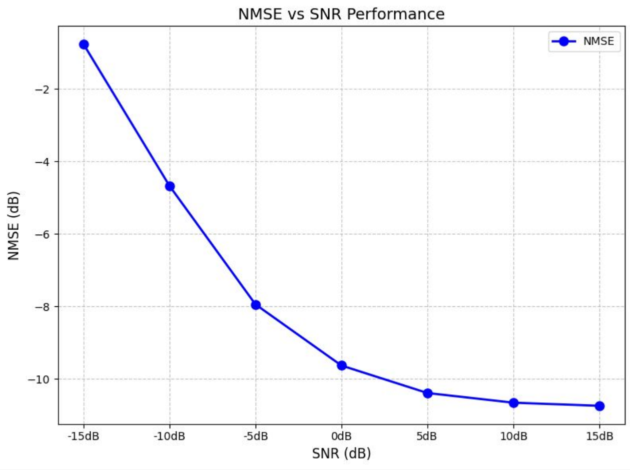
#4 Spike and Slab Prior

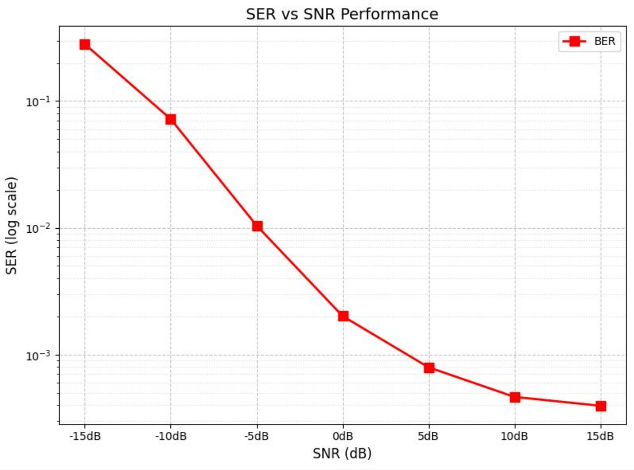



```
data {
  int<lower=1> N;
  int<lower=1> M;
  array[N] int<lower=0, upper=1> Y;
  matrix[N, M] H;
}
parameters {
  vector[M] x;
  vector<lower=0, upper=1>[M] theta;
  vector<lower=1e-6>[M] alpha;
}
transformed parameters {
  vector[M] alpha_inv = sqrt(1/alpha);
  vector[N] logits = H * x;
}
model {
  alpha ~ gamma(1e-3, 1e-3);
  for (m in 1:M) {
    target += log(1 - theta[m]) + normal_lpdf(x[m] | 0,
alpha_inv[m]);
    target += log(theta[m]) + normal_lpdf(x[m] | 0, 1);
  }
  theta ~ beta(1,1);
  Y ~ bernoulli_logit(logits);
}
```

Joint CEDD (QAM Signal)
ADVI

#RESULTS





FIXED PARAMS

- $M = 20$
- $N_T = 128$
- $T_p = 80$
- $T_d = 200$

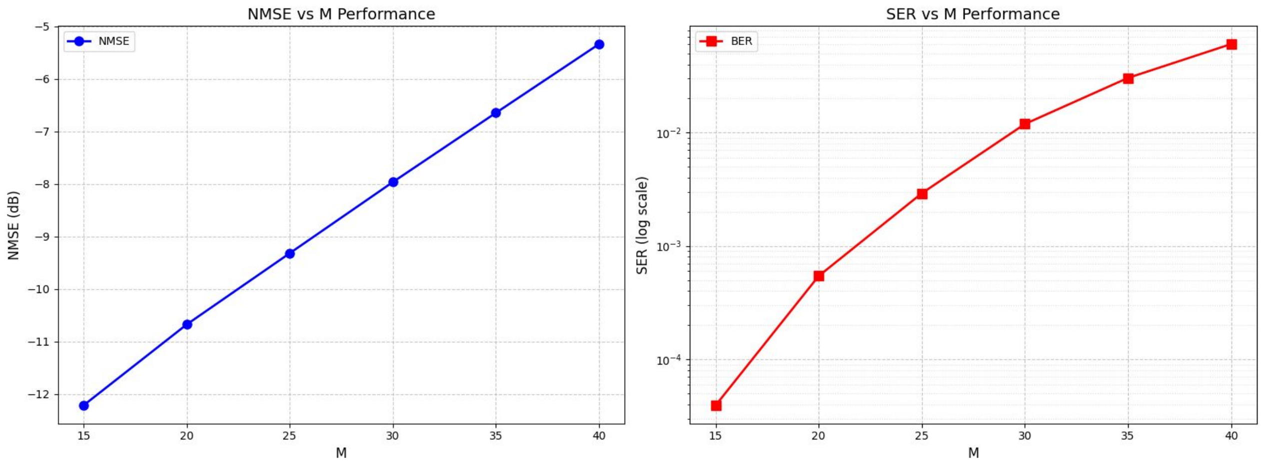
VARIABLE PARAM

- SNR(dB) = -15:15:5

DONE = True

```
NMSE_Means =
{'-15dB': -0.7603058714894002,
  '-10dB': -4.674833033425079,
  '-5dB': -7.947453473907131,
  '0dB': -9.635551188189567,
  '5dB': -10.39496506533456,
  '10dB': -10.66201555308801,
  '15dB': -10.747631241721129}
```

```
SER_Means =
{'-15dB': 0.28130000000000005,
  '-10dB': 0.0721,
  '-5dB': 0.010416666666666666,
  '0dB': 0.0020131578947368426,
  '5dB': 0.00079375,
  '10dB': 0.00046341463414634155,
  '15dB': 0.00039423076923076933}
```



FIXED PARAMS

- **SNR(dB) = 10**
- **N_T = 128**
- **T_p = 80**
- **T_d = 200**

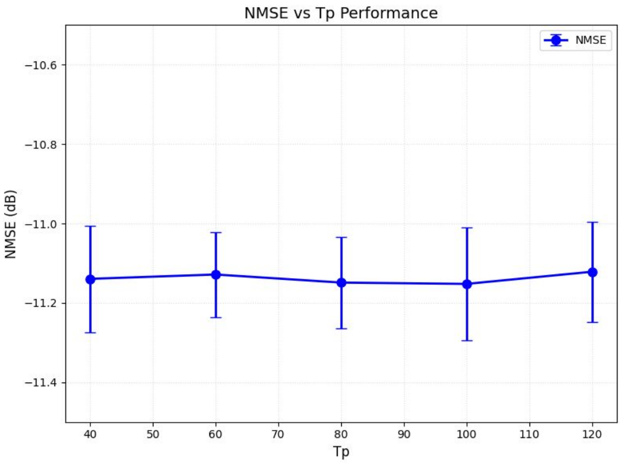
VARIABLE PARAM

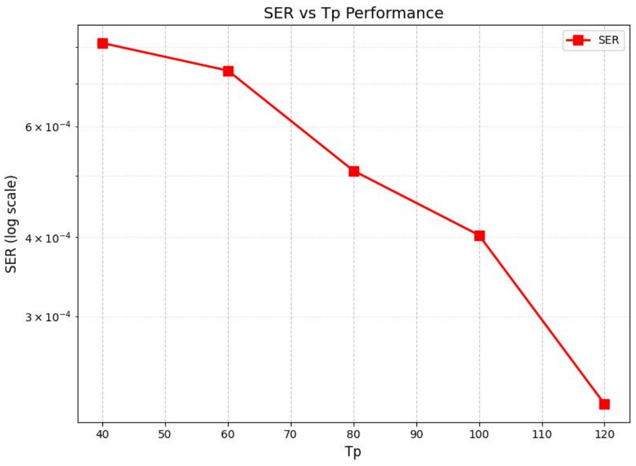
- **M = 15:40:5**

DONE = True

```
NMSE_Means =
{15: -12.213640841944153,
  20: -10.674307748775732,
  25: -9.31654630823653,
  30: -7.961907129861372,
  35: -6.643087361851371,
  40: -5.333969446637535}
```

```
SER_Means =
{15: 3.9215686274505875e-05,
  20: 0.0005431034482758620,
  25: 0.002914285714285714 3,
  30: 0.011908045977011462,
  35: 0.030367346938775464,
  40: 0.06039285714285714 4}
```





FIXED PARAMS

- SNR(dB) = 10
- M = 20
- N_T = 128
- T_d = 200

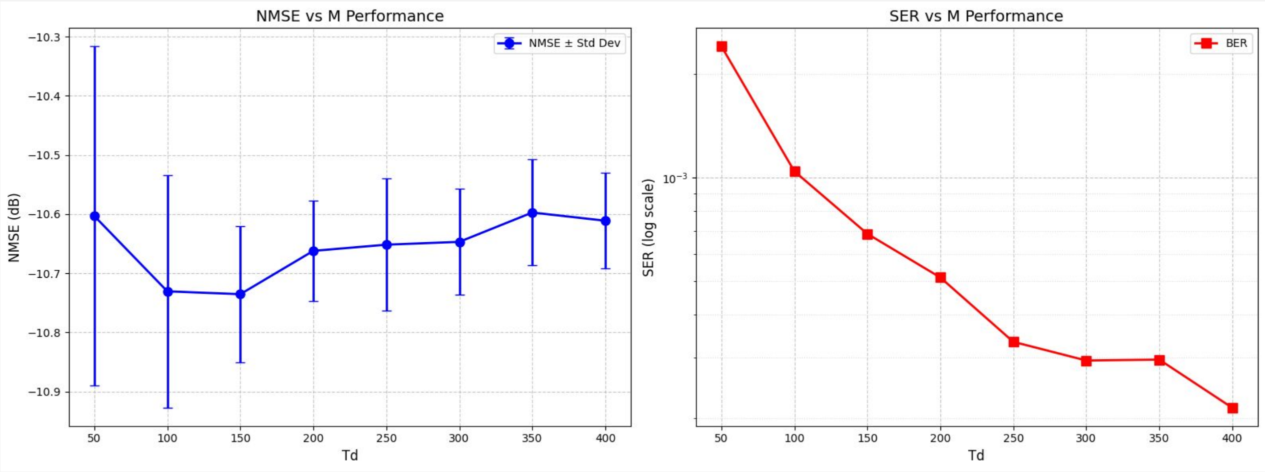
VARIABLE PARAM

- T_p = 40:120:20

DONE = False

```
NMSE_Means =
{40: -11.139461768975295,
  60: -11.12852102366174,
  80: -11.148851011167851,
  100: -11.152220558513124,
  120: -11.121575169194818}
```

```
SER_Means =
{40: 0.0008125000000000001,
  60: 0.0007348484848484849,
  80: 0.00051,
  100: 0.00040322580645161296,
  120: 0.0002177419354838710}
```



FIXED PARAMS

- **SNR(dB) = 10**
- **M = 20**
- **N_T = 128**
- **T_P = 80**

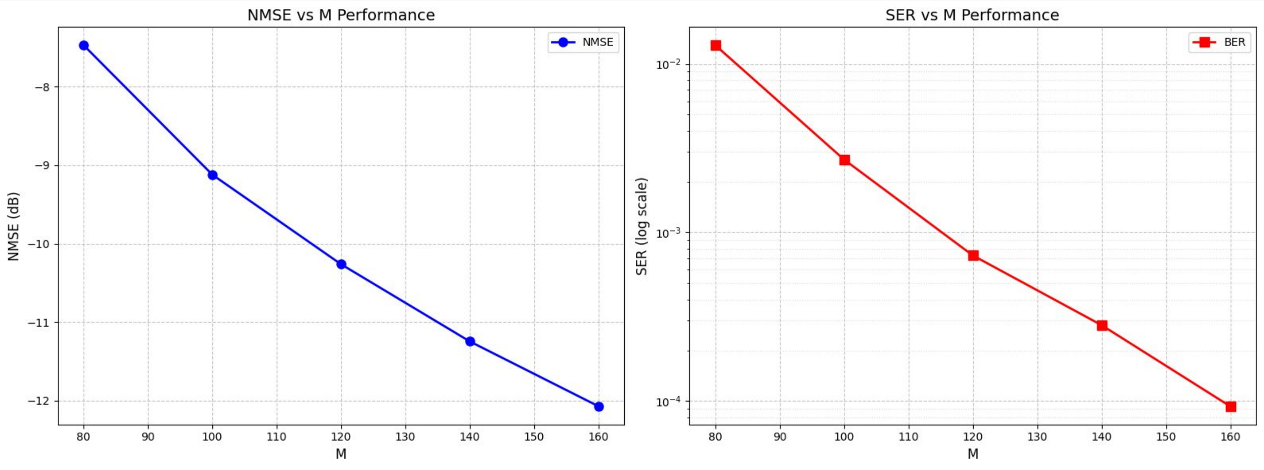
VARIABLE PARAM

- **T_d = 50:400:50**

DONE = True

```
NMSE_Means =
{50: -10.603215147525685,
  100: -10.730484943970025,
  150: -10.735331959297419,
  200: -10.66213450262405,
  250: -10.651680861219141,
  300: -10.646915900197357,
  350: -10.597318572011277,
  400: -10.611180282878346}
```

```
SER_Means =
{50: 0.0024166666666666664,
  100: 0.0010454545454545456,
  150: 0.0006874999999999975,
  200: 0.0005131578947368422,
  250: 0.0003333333333333333,
  300: 0.0002941176470587882,
  350: 0.0002959183673468996,
  400: 0.00021428571428571436}
```



FIXED PARAMS

- **SNR(dB) = 10**
- **M = 20**
- **T_P = 80**
- **T_d = 200**

VARIABLE PARAM

- **N_T = 80:160:20**

DONE = True

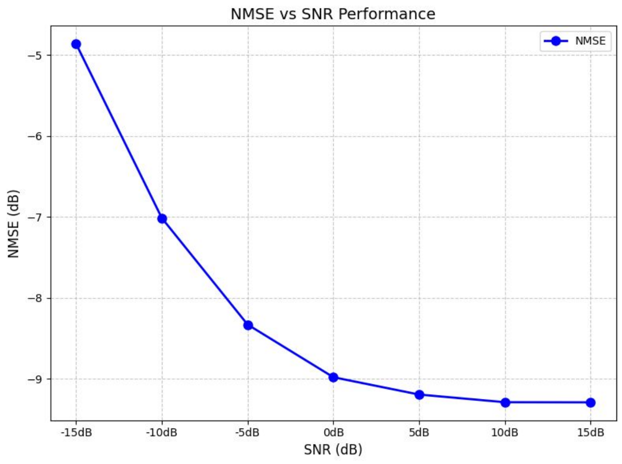
```
NMSE_Means =
{80: -7.47102150803973,
  100: -9.121251600905753,
  120: -10.262386655347425,
  140: -11.24588301144952,
  160: -12.072205639770708}
```

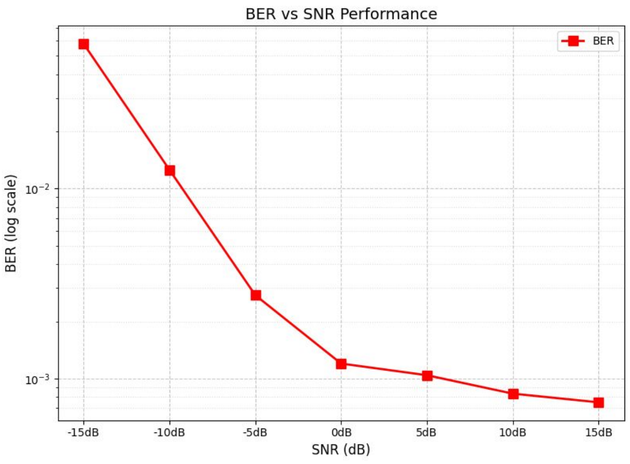
```
SER_Means =
{80: 0.012916666666666667,
  100: 0.0027083333333333334,
  120: 0.0007307692307692309,
  140: 0.00028197674418604666,
  160: 9.294871794871796e-05}
```

Joint CEDD (QAM Signal)

PVI

#RESULTS





FIXED PARAMS

- **$M = 20$**
- **$N = 4$**
- **$L = 32$**
- **$N_T(N*L) = 128$**
- **$T_p = 80$**
- **$T_d = 200$**

VARIABLE PARAM

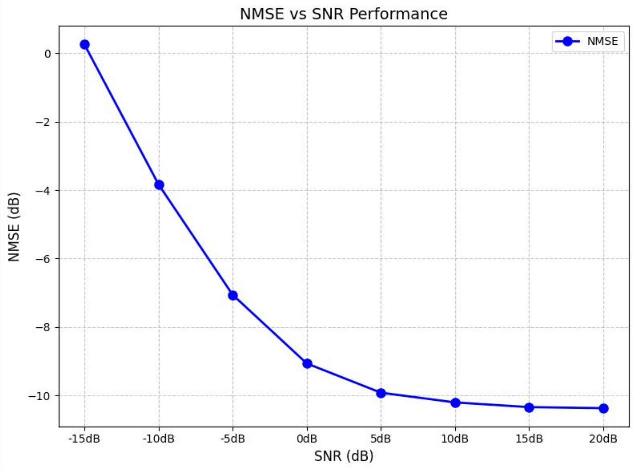
- **SNR(dB) = -15:15:5**

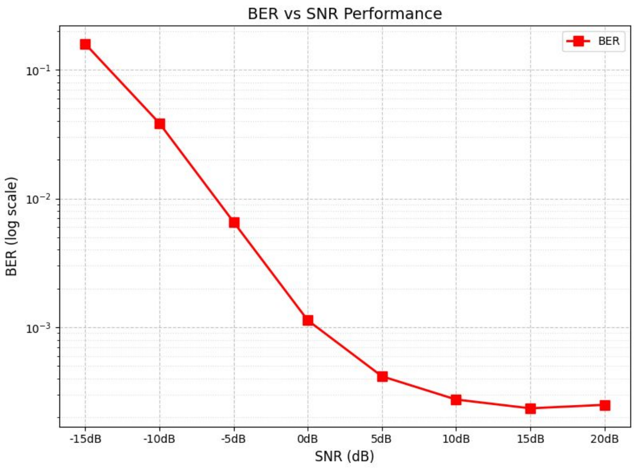
Seq_DONE = True

Sync_DONE = False

```
Seq_NMSE_Means =
{'-15dB': -4.859897294059982,
  '-10dB': -7.014276173033149,
  '-5dB': -8.330474140747567,
  '0dB': -8.981455368343433,
  '5dB': -9.195674698770317,
  '10dB': -9.290274615426473,
  '15dB': -9.291771645207321}
Sync_NMSE_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
```

```
Seq_BER_Means =
{'-15dB': 0.05787499999999996,
  '-10dB': 0.0125625,
  '-5dB': 0.00275,
  '0dB': 0.0012000000000000001,
  '5dB': 0.0010416666666666667,
  '10dB': 0.0008333333333333334,
  '15dB': 0.0007499999999999999}
Sync_BER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```





FIXED PARAMS

- **SNR(dB) = 10**
- **N = 4**
- **L = 32**
- **N_T(N*L) = 128**
- **T_p = 80**
- **T_d = 200**

VARIABLE PARAM

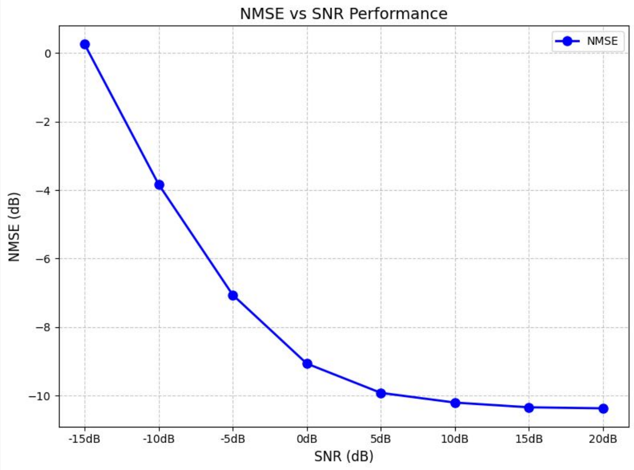
- **M = 15:40:5**

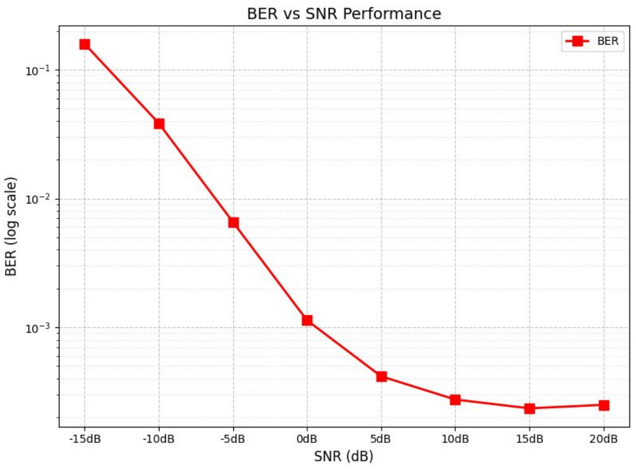
Seq_DONE = False

Sync_DONE = False

```
Seq_NMSE_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
Sync_NMSE_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
```

```
Seq_BER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
Sync_BER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
```






```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```

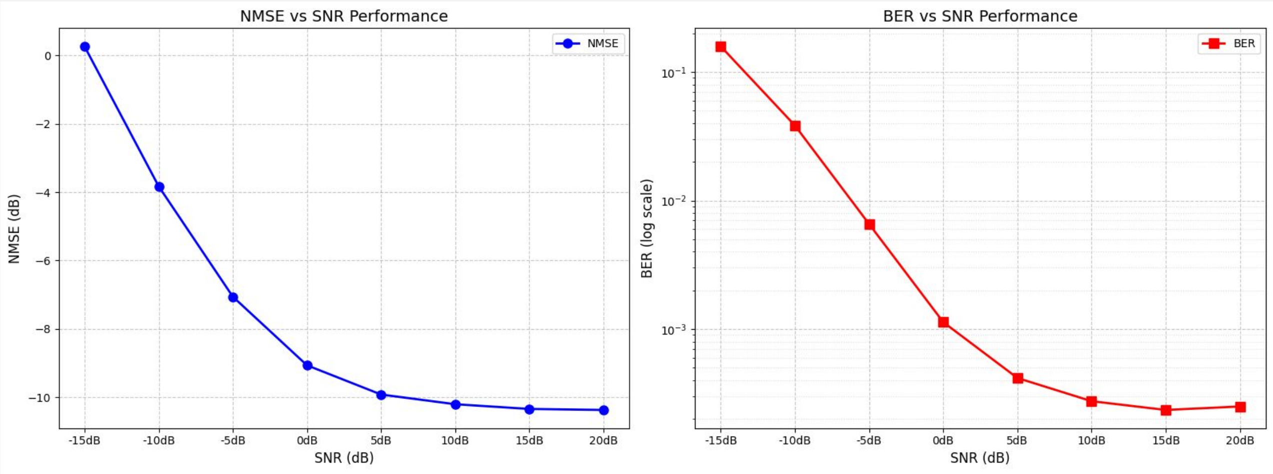
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}

FIXED PARAMS
- **SNR(dB) = 10**
- **M = 20**
- **N = 4**
- **L = 32**
- **N_T(N*L) = 128**
- **T_d = 200**

VARIABLE PARAM
- **T_p = 40:120:20**

Seq_DONE = False

Sync_DONE = False




```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```

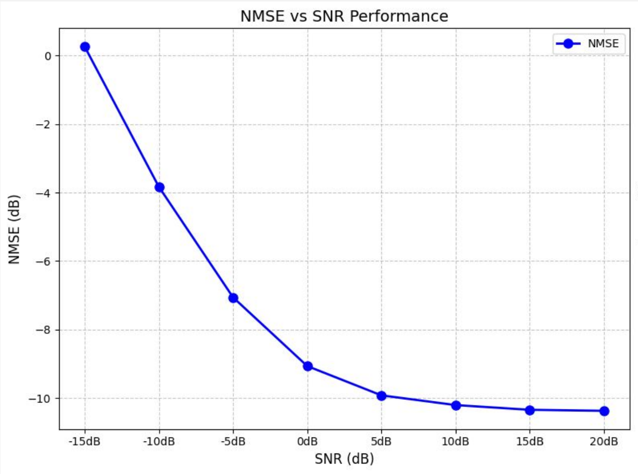

```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```

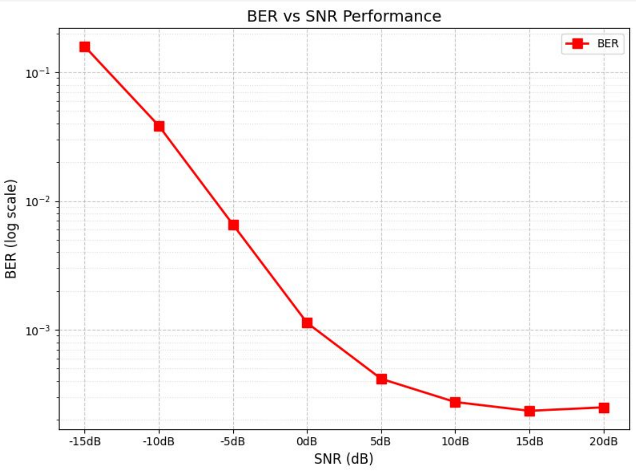

FIXED PARAMS
- SNR(dB) = 10
- M = 20
- N = 4
- L = 32
- N_T(N*L) = 128
- T_P = 80

VARIABLE PARAM
- T_d = 160:300:20

Seq_DONE = False

Sync_DONE = False






```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```


```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```


FIXED PARAMS

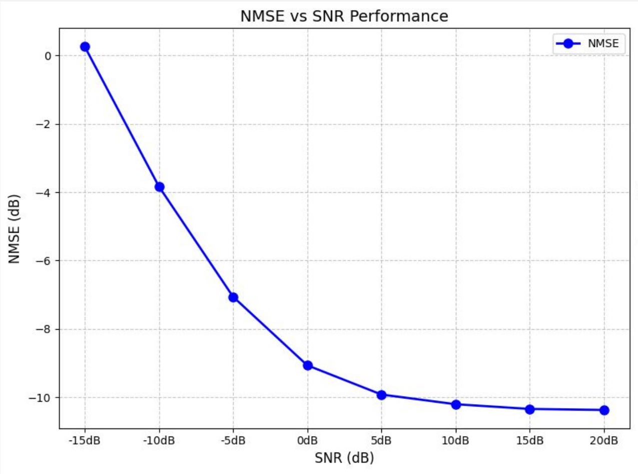
- **SNR(dB) = 10**
- **M = 20**
- **N = 4**
- **T_P = 80**
- **T_d = 200**

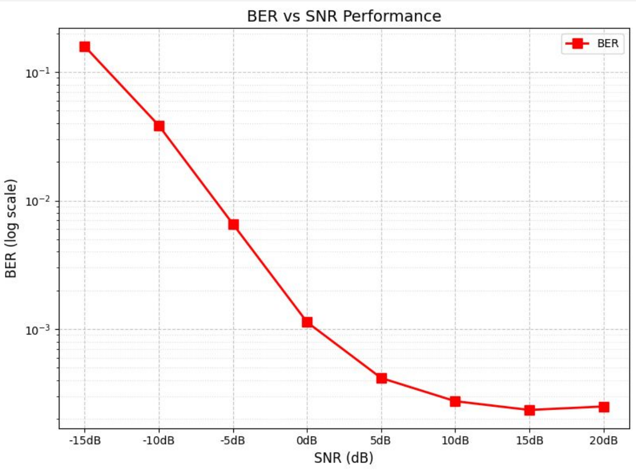
VARIABLE PARAM

- **L = 30:50:4**
- **N_T = N * L**

Seq_DONE = False

Sync_DONE = False






```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```


```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```


FIXED PARAMS

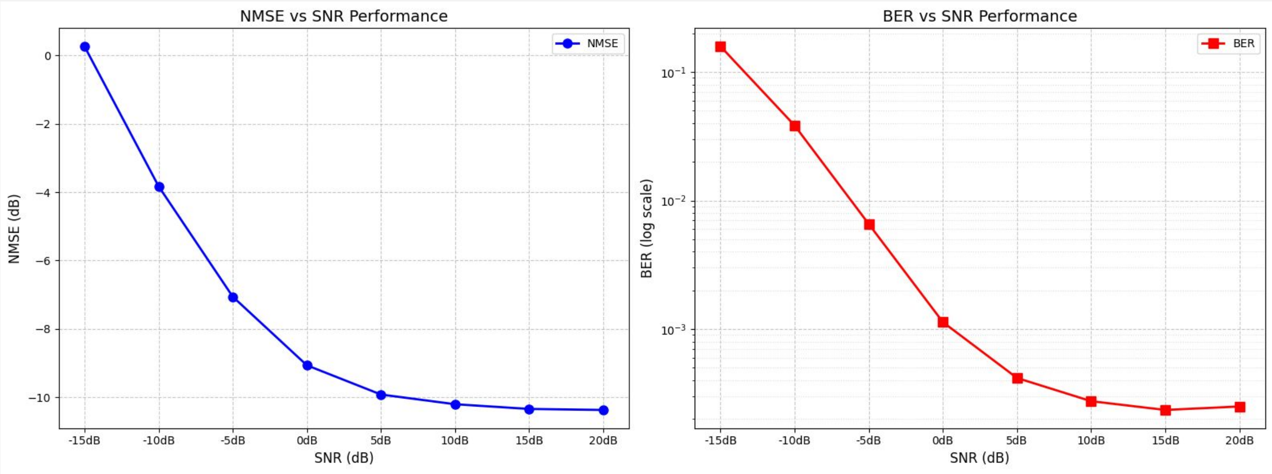
- **SNR(dB) = 10**
- **M = 20**
- **L = 32**
- **T_p = 80**
- **T_d = 200**

VARIABLE PARAM

- **N = 4:12:1**
- **N_T = N * L**

Seq_DONE = False

Sync_DONE = False




```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.002168319169027351 7}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```


```
Seq_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
Sync_SER_Means =
{'-15dB': 0.35761449480642116,
  '-10dB': 0.1416283050047214,
  '-5dB': 0.04049043909348439,
  '0dB': 0.011150849858356909,
  '5dB': 0.004284702549575039,
  '10dB': 0.002644003777148219,
  '15dB': 0.0021683191690273517}
```


FIXED PARAMS
- **SNR(dB) = 10**
- **M = 20**
- **N_T(N*L) = 128**
- **T_p = 80**
- **T_d = 200**

VARIABLE PARAM
- **N = [1,2,4,8,16,32]**
- **L = [128,64,32,16,8,4]**

Seq_DONE = False

Sync_DONE = False

Massive MIMO

Activity Detection + Channel Estimation

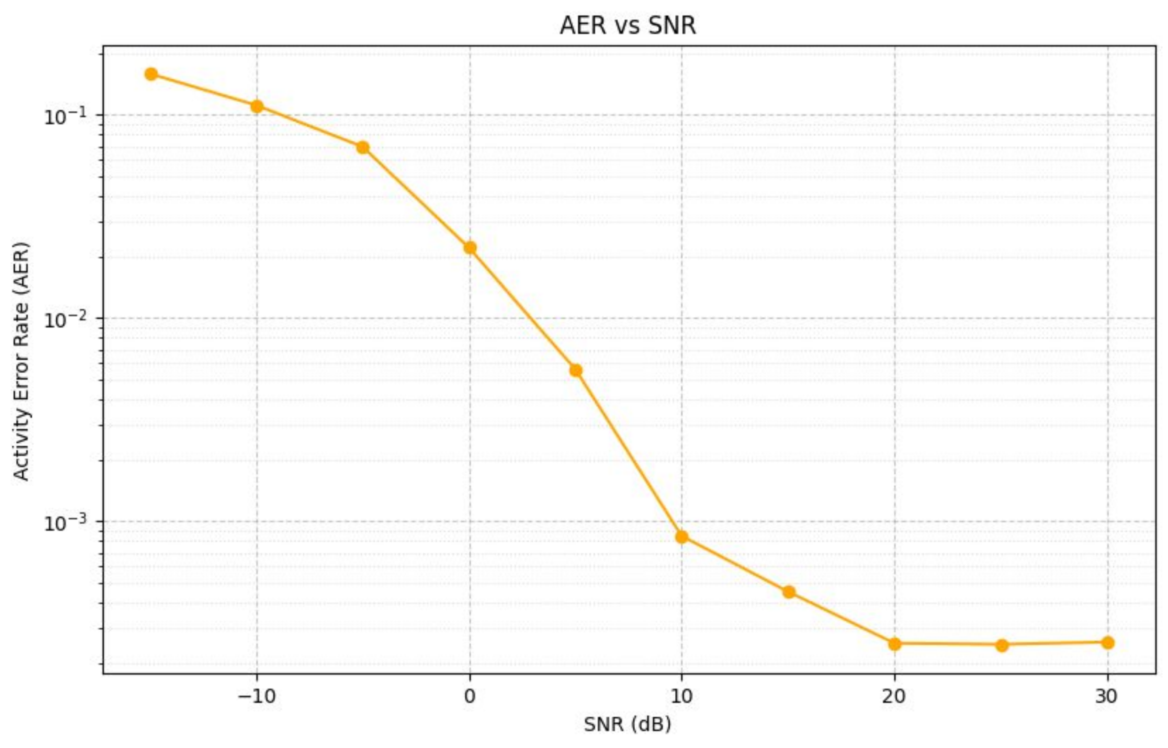
Model Specification

- Y = Hx + noise
- Y and noise are NxTp matrix, H is NxM matrix and H ~ N(0,1).
- X is a sparse matrix of size MxTp, with each column having only 10% non-zero values. These are equi-distributed +1/-1. Rest 90% values re 0.
- We proceed by taking into account the activity of nodes - which is a diagonal matrix with +1, 0, -1 values.
- Hare, we do channel estimation and activity detection.

STAN Model

```
data {
  int<lower=1> N;
  int<lower=1> M;
  int<lower=1> Tp;
  array[N, Tp] int Y;
  matrix[M, Tp] x;
  real sigma;
}
parameters {
  matrix[N, M] H;
  vector[M] activity;
}
model {
  for(n in 1:N){
    H[n] ~ normal(0,1);
  }
  activity ~ double_exponential(0, 1);
  for(tp in 1:Tp){
    Y[:, tp] ~ bernoulli_logit(H * diag_matrix(activity) * x[:, tp]);
  }
}
```

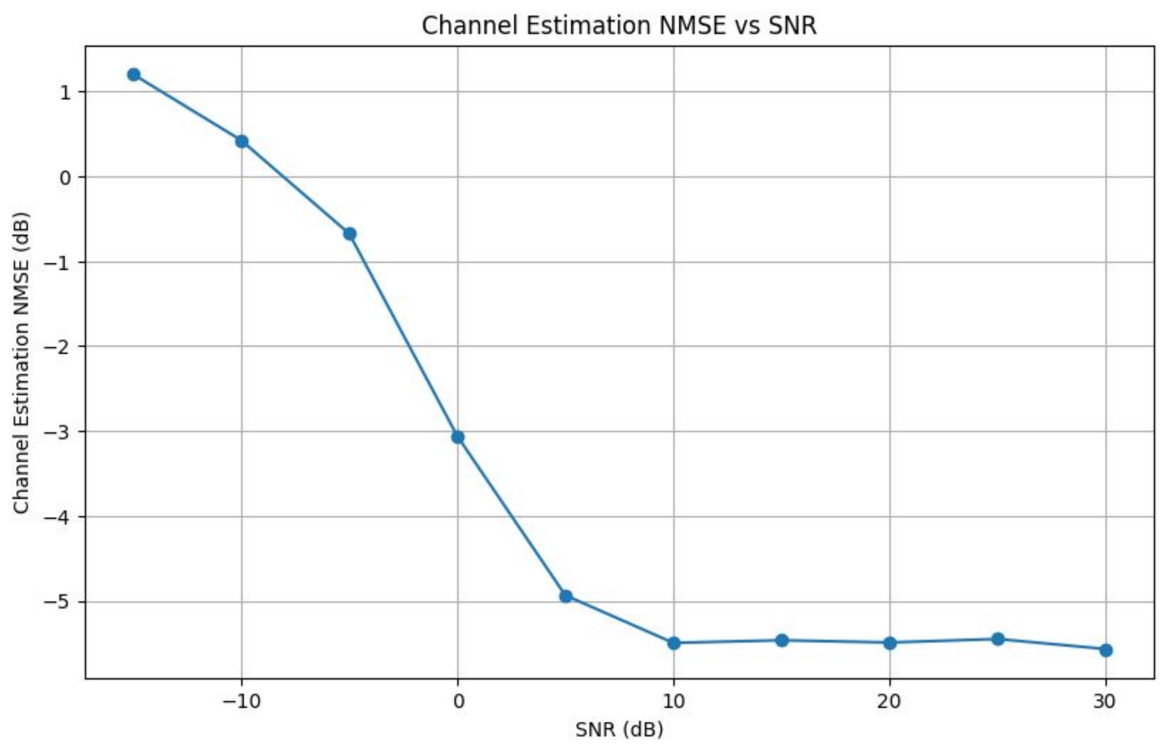
Results: Activity Error Rate(AER)




```
# Real system
params = {
    'N': 32,
    'M': 50,
    'Tp': 20,
}
```

```
Means =
{-15: 0.15888888888888889,
 -10: 0.11130434782608696,
 -5: 0.06956521739130435,
 0: 0.022178217821782218,
 5: 0.0056000000000000001,
 10: 0.0008474576271186442,
 15: 0.0004511278195488722,
 20: 0.00025157232704402514,
 25: 0.0002484472049689441,
 30: 0.0002547770700636943}
```

Results: Channel NMSE

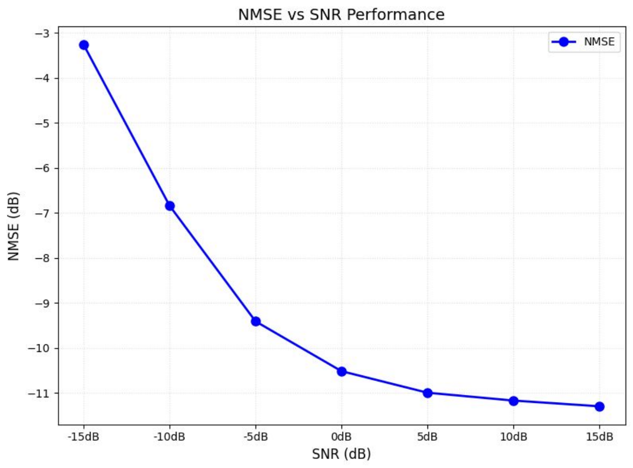


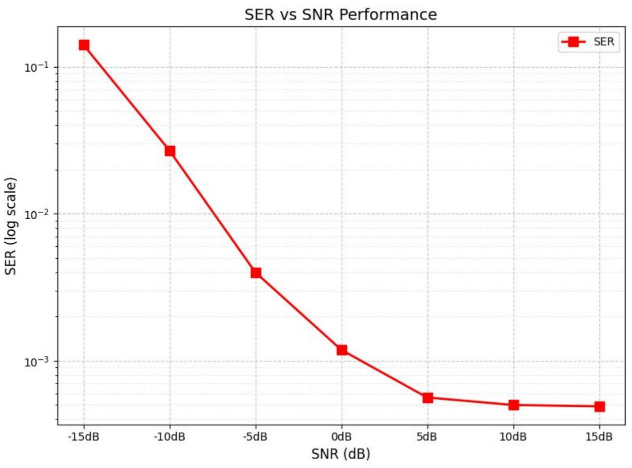

```
# Real system
params = {
    'N': 32,
    'M': 50,
    'Tp': 20,
}
```

```
Means =
{-15: 1.2035316440291224,
 -10: 0.42603632918596235,
 -5: -0.6746170164115171,
 0: -3.06000301048139,
 5: -4.936468543170566,
 10: -5.496965866101205,
 15: -5.4657569555788035,
 20: -5.492109246712657,
 25: -5.450827278207181,
 30: -5.569071640408481}
```

STAN Model - Complex Signal

```
data {
  int<lower=1> N;
  int<lower=1> M;
  int<lower=1> Tp;
  array[N, Tp] int Y;
  matrix[M, Tp] x;
  real sigma;
}
parameters {
  matrix[N, M] H;
  vector[M %/% 2] activity;
}
model {
  // Priors
  for(n in 1:N){
    H[n] ~ normal(0,1);
  }
  activity ~ double_exponential(0, 1);
  // Likelihood
  for(tp in 1:Tp){
    Y[:, tp] ~ bernoulli_logit(H * diag_matrix(append_row(activity,activity)) * x[:, tp]);
  }
}
```





 # QAM Symbols

FIXED PARAMS

- **N = 64**
- **M = 100**
- **T_p = 40**
- **T_d = 100**
- **Sparsity = 10%**

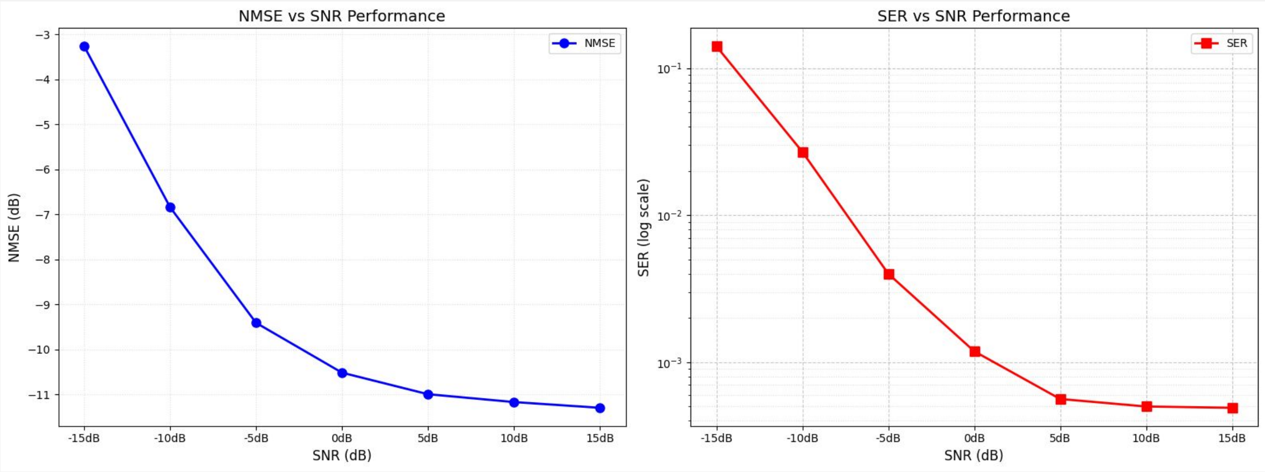
VARIABLE PARAM

- **SNR(dB) = -15:30:5**

DONE = False

```
NMSE_Means =
{'-15dB': -3.260166191523126,
  '-10dB': -6.840488027886185,
  '-5dB': -9.411511734776637,
  '0dB': -10.519929337058343,
  '5dB': -10.998835440426967,
  '10dB': -11.173559976315325,
  '15dB': -11.300944061780347}
```

```
AER_Means =
{'-15dB': 0.1411,
  '-10dB': 0.027,
  '-5dB': 0.004,
  '0dB': 0.0011875,
  '5dB': 0.0005625000000000001,
  '10dB': 0.0005,
  '15dB': 0.00049}
```



 # QAM Symbols

FIXED PARAMS

- **N = 64**
- **M = 100**
- **T_p = 40**
- **T_d = 100**
- **SNR(dB) = 10dB**

VARIABLE PARAM

- **Sparsity(%) = 10:50:10**

DONE = False

```
NMSE_Means =
{'-15dB': -3.260166191523126,
  '-10dB': -6.840488027886185,
  '-5dB': -9.411511734776637,
  '0dB': -10.519929337058343,
  '5dB': -10.998835440426967,
  '10dB': -11.173559976315325,
  '15dB': -11.300944061780347}
```

```
AER_Means =
{'-15dB': 0.1411,
  '-10dB': 0.027,
  '-5dB': 0.004,
  '0dB': 0.0011875,
  '5dB': 0.0005625000000000001,
  '10dB': 0.0005,
  '15dB': 0.00049}
```

tq :)