SSY316 Assignment 4

Edoardo Mangia
edoardom@student.chalmers.se

Nuree Kim
nuree@student.chalmers.se

December 2024

Contents

1 Exercise 1

Prior for a:

$$P(a = \text{true}) = 0.3, \quad P(a = \text{false}) = 0.7.$$

Factor $f(a, b)$:

$$f(\text{true}, \text{true}) = 0.9, \quad f(\text{true}, \text{false}) = 0.1, \quad f(\text{false}, \text{true}) = 0.2, \quad f(\text{false}, \text{false}) = 0.8.$$

The observation is that $b = \text{true}$.

Using the factor table, the likelihood of $b = \text{true}$ for each value of a is:

$$P(b = \text{true} \mid a = \text{true}) = f(\text{true}, \text{true}) = 0.9,$$

$$P(b = \text{true} \mid a = \text{false}) = f(\text{false}, \text{true}) = 0.2.$$

Using the prior $P(a)$ and the likelihood $P(b \mid a)$:

$$P(a = \text{true}, b = \text{true}) = P(a = \text{true}) \cdot P(b = \text{true} \mid a = \text{true}),$$

$$P(a = \text{true}, b = \text{true}) = 0.3 \cdot 0.9 = 0.27.$$

Similarly:

$$P(a = \text{false}, b = \text{true}) = P(a = \text{false}) \cdot P(b = \text{true} \mid a = \text{false}),$$

$$P(a = \text{false}, b = \text{true}) = 0.7 \cdot 0.2 = 0.14.$$

Summing over all possible values of a:

$$P(b = \text{true}) = P(a = \text{true}, b = \text{true}) + P(a = \text{false}, b = \text{true}),$$

$$P(b = \text{true}) = 0.27 + 0.14 = 0.41.$$

Using Bayes' Rule:

$$P(a \mid b = \text{true}) = \frac{P(a, b = \text{true})}{P(b = \text{true})}.$$

For $a = \text{true}$:

$$P(a = \text{true} \mid b = \text{true}) = \frac{0.27}{0.41} \approx 0.6585.$$

For $a = \text{false}$:

$$P(a = \text{false} \mid b = \text{true}) = \frac{0.14}{0.41} \approx 0.3415.$$

The posterior distribution of a given $b = \text{true}$ is:

$$P(a = \text{true} \mid b = \text{true}) \approx 0.6585,$$

$$P(a = \text{false} \mid b = \text{true}) \approx 0.3415.$$

2 Exercise 2

The factor graph involves two continuous scalar random variables a and b with the following factor functions:

$$f_1(a) = \mathcal{N}(a; \mu_1, \sigma_1^2) \quad \text{and} \quad f_2(a, b) = \mathcal{N}(b; \alpha a, \sigma_2^2)$$

Q1 - Compute the Marginal Distribution of a

The marginal distribution of a is given by:

$$p(a) \propto f_1(a) \int f_2(a, b)\, db.$$

Substituting the given factors:

$$f_1(a) = \mathcal{N}(a; \mu_1, \sigma_1^2), \quad f_2(a, b) = \mathcal{N}(b; \alpha a, \sigma_2^2),$$

we can simplify the integral. Since $f_2(a, b)$ integrates to 1 with respect to b, we get:

$$p(a) = \mathcal{N}(a; \mu_1, \sigma_1^2).$$

Thus, the marginal distribution of a is determined entirely by $f_1(a)$.

Q2 - Compute the Marginal Distribution of b

The marginal distribution of b is given by:

$$p(b) \propto \int f_2(a,b) f_1(a)\, da.$$

Substituting the Gaussian forms:

$$f_1(a) = \mathcal{N}(a; \mu_1, \sigma_1^2), \quad f_2(a,b) = \mathcal{N}(b; \alpha a, \sigma_2^2),$$

we combine the terms:

$$f_1(a) f_2(a,b) \propto \exp\left(-\frac{(a-\mu_1)^2}{2\sigma_1^2} - \frac{(b-\alpha a)^2}{2\sigma_2^2}\right).$$

Expanding $(b-\alpha a)^2$:

$$(b-\alpha a)^2 = b^2 - 2\alpha ab + \alpha^2 a^2.$$

The quadratic terms in a are:

$$-\frac{a^2}{2\sigma_1^2} - \frac{\alpha^2 a^2}{2\sigma_2^2} + \frac{2\alpha ab}{2\sigma_2^2} - \frac{2\mu_1 a}{2\sigma_1^2}.$$

Factoring a^2:

$$-\frac{a^2}{2}\left(\frac{1}{\sigma_1^2} + \frac{\alpha^2}{\sigma_2^2}\right) + a\left(\frac{\alpha b}{\sigma_2^2} + \frac{\mu_1}{\sigma_1^2}\right).$$

Completing the square for a, we obtain:

$$-\frac{(a-\mu')^2}{2\sigma'^2},$$

where:

$$\mu' = \frac{\frac{\alpha b}{\sigma_2^2} + \frac{\mu_1}{\sigma_1^2}}{\frac{1}{\sigma_1^2} + \frac{\alpha^2}{\sigma_2^2}}, \quad \sigma'^2 = \frac{1}{\frac{1}{\sigma_1^2} + \frac{\alpha^2}{\sigma_2^2}}.$$

Integrating over a, the resulting marginal distribution of b is:

$$p(b) = \mathcal{N}(b; \alpha\mu_1, \alpha^2\sigma_1^2 + \sigma_2^2).$$

3 Exercise 3

Q1 - Conditional Distributions and Bayesian Network

The variables in this model are:

- Y (quality of steel), which can take two values: high-quality steel ($Y = 1$) or low-quality steel ($Y = 0$).

- C (clips produced), which follows a Poisson distribution with parameter λ.

- P (pins produced), which also follows a Poisson distribution with parameter λ.

The conditional distributions are:

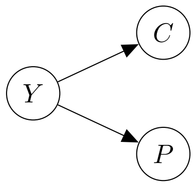
$$p(C|Y=1) = \frac{10^C \exp(-10)}{C!}, \quad p(P|Y=1) = \frac{10^P \exp(-10)}{P!}$$

$$p(C|Y=0) = \frac{7^C \exp(-7)}{C!}, \quad p(P|Y=0) = \frac{7^P \exp(-7)}{P!}$$

The prior distribution for Y is:

$$p(Y=1) = 0.25, \quad p(Y=0) = 0.75$$

The Bayesian network for this model is:



The production of the clips and pins is dependent on the quality of the steel.

Q2 - Factor Graph Transformation

To transform the Bayesian network into a factor graph, we represent the joint distribution as a product of factors corresponding to each random variable and its dependencies.

The factor graph is composed of the following factors:

$$f_Y(Y) = p(Y)$$

$$f_C(C,Y) = p(C|Y)$$

$$f_P(P,Y) = p(P|Y)$$

The factor graph would have one factor node for each of these conditional distributions and a variable node for each random variable. An edge connects a variable node to the factor node corresponding to its conditional distribution.

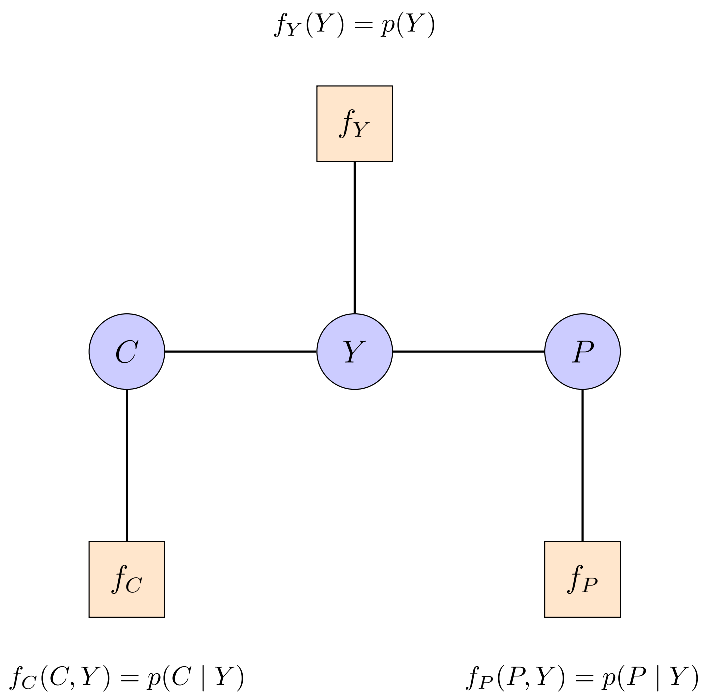


Figure 1: Factor graph representation of the Bayesian network.

Q3 - Message Passing for High-Quality Steel Probability

To compute the probability that the company was using high-quality steel, we use the message passing algorithm in the factor graph. The goal is to compute $p(Y = 1|C = 10, P = 8)$, the probability that the steel quality was high given the production of 10 clips and 8 pins.

Assuming that the number of clips C and pins P produced are independent given the quality of steel Y, we have:

$$p(C = 10, P = 8 \mid Y) = p(C = 10 \mid Y) \cdot p(P = 8 \mid Y)$$

For High-Quality Steel ($Y = 1$):

$$\lambda_{\text{high}} = 10$$

$$p(C = 10 \mid Y = 1) = \frac{10^{10}e^{-10}}{10!} \approx 0.1251$$

$$p(P = 8 \mid Y = 1) = \frac{10^8 e^{-10}}{8!} \approx 0.1126$$

$$p(C = 10, P = 8 \mid Y = 1) = p(C = 10 \mid Y = 1) \times p(P = 8 \mid Y = 1) \approx 0.1251 \times 0.1126 = 0.01407$$

For Low-Quality Steel ($Y = 0$):

$$\lambda_{\text{low}} = 7$$

$$p(C = 10 \mid Y = 0) = \frac{7^{10} e^{-7}}{10!} \approx 0.0708$$

$$p(P = 8 \mid Y = 0) = \frac{7^8 e^{-7}}{8!} \approx 0.1302$$

$$p(C = 10, P = 8 \mid Y = 0) = p(C = 10 \mid Y = 0) \times p(P = 8 \mid Y = 0) \approx 0.0708 \times 0.1302 = 0.00922$$

Incorporating the Priors:

$$p(Y = 1) = 0.25$$

$$p(Y = 0) = 0.75$$

Computing the Joint Probabilities:

$$p(Y = 1, C = 10, P = 8) = p(Y = 1) \times p(C = 10, P = 8 \mid Y = 1)$$

$$= 0.25 \times 0.01407 = 0.0035175$$

$$p(Y = 0, C = 10, P = 8) = p(Y = 0) \times p(C = 10, P = 8 \mid Y = 0)$$

$$= 0.75 \times 0.00922 = 0.006915$$

Apply Bayes' Theorem

$$p(Y = 1 \mid C = 10, P = 8) = \frac{p(Y = 1, C = 10, P = 8)}{p(Y = 1, C = 10, P = 8) + p(Y = 0, C = 10, P = 8)}$$

$$= \frac{0.0035175}{0.0035175 + 0.006915} \approx 0.3373$$

The posterior probability that the company was using high-quality steel given that 10 clips and 8 pins were produced is approximately 33.73%.

$$p(Y = 1 \mid C = 10, P = 8) \approx 0.3373$$

Q4 - Monte Carlo Simulation Program

To verify the calculated probability using Monte Carlo simulation, we can write a Python program as follows:

```python
# Parameters
lambda_high = 10
lambda_low = 7
p_high_quality = 0.25
num_simulations = 100000

# Simulate the production process
high_quality_steel = np.random.rand(num_simulations) <
    p_high_quality

clips_produced = np.where(high_quality_steel,
                          np.random.poisson(lambda_high,
                              num_simulations),
                          np.random.poisson(lambda_low,
                              num_simulations))

pins_produced = np.where(high_quality_steel,
                         np.random.poisson(lambda_high,
                             num_simulations),
                         np.random.poisson(lambda_low,
                             num_simulations))

# Count the simulations where clips = 10 and pins = 8
mask = (clips_produced == 10) & (pins_produced == 8)
high_quality_count = np.sum(mask & high_quality_steel)

# Estimate the probability of high-quality steel
total_count = np.sum(mask)
probability_high_quality = high_quality_count / total_count if
    total_count > 0 else 0

print(f"Estimated probability of high-quality steel:
    {probability_high_quality:.4f}")
```

Listing 1: Monte Carlo Simulation for High-Quality Steel Probability

```
    Estimated probability of high-quality steel: 0.3256
```

This program will simulate the production process for a large number of days and estimate the probability that the steel quality was high, given that the company produced 10 clips and 8 pins.

4 Exercise 4

Q1 - Full Joint Probability Distribution

The joint probability distribution $P(L, S, G, D, I)$ is expressed as:

$$P(L, S, G, D, I) = P(L \mid G) \cdot P(S \mid I) \cdot P(G \mid D, I) \cdot P(D) \cdot P(I)$$

where:

- $P(L \mid G)$: Conditional probability of the letter given the grade.
- $P(S \mid I)$: Conditional probability of SAT score given intelligence.
- $P(G \mid D, I)$: Conditional probability of the grade given difficulty and intelligence.
- $P(D)$: Marginal probability of the difficulty.
- $P(I)$: Marginal probability of intelligence.

Q2 - Conditional Independence

Two sets of conditionally independent variables given another variable:

1. Difficulty (D) is conditionally independent of SAT (S) given Intelligence (I).
2. Letter (L) is conditionally independent of Difficulty (D), Intelligence (I), and SAT (S) given Grade (G).

Q3 - Probability Calculation $P(I = High \mid G = C)$

Using Bayes' theorem:

$$P(I = High \mid G = C) = \frac{P(G = C \mid I = High) \cdot P(I = High)}{P(G = C)}$$

where:

- $P(G = C \mid I = High)$ is derived from $P(G \mid D, I)$.
- $P(I = High)$: Marginal probability of intelligence.
- $P(G = C) : \sum_{D,I} P(G = C \mid D, I) \cdot P(D) \cdot P(I)$

Q4 - Probability Calculation $P(I = High \mid G = C, S = Good)$

Using Bayes' theorem and conditional independence:

$$P(I = High \mid G = C, S = Good) = \frac{P(S = Good \mid I = High) \cdot P(G = C \mid I = High) \cdot P(I = High)}{P(G = C, S = Good)}$$

where:

- $P(S = Good \mid I = High)$: From $P(S \mid I)$.
- $P(G = C \mid I = High)$: From $P(G \mid D, I)$.
- $P(G = C, S = Good) : \sum_{I,D} P(S = Good \mid I) \cdot P(G = C \mid D, I) \cdot P(D) \cdot P(I)$

Q5 - Probability Calculation $P(D = Hard \mid G = C, S = Good)$

Using Bayes' theorem:

$$P(D = Hard \mid G = C, S = Good) = \frac{P(G = C, S = Good \mid D = Hard) \cdot P(D = Hard)}{P(G = C, S = Good)}$$

where:

- $P(G = C, S = Good \mid D = Hard) = \sum_I P(S = Good \mid I) \cdot P(G = C \mid D = Hard, I) \cdot P(I)$
- $P(D = Hard)$: Marginal probability of difficulty.
- $P(G = C, S = Good)$: Already calculated in Q4.

This shows how $S = Good$ and $G = C$ increase belief about $D = Hard$.

Q6 - Causality vs. Correlation

Causality implies a direct effect on a factor. For instance, Intelligence directly affects SAT scores because smarter students tend to perform better on SATs.

Correlation implies a statistical association without causation. For instance, SAT scores and Grades may be correlated because both are influenced by Intelligence, but one does not directly cause the other.

This distinction can be illustrated in the network, where I causes both G and S, but S does not cause G.

5 Exercise 5

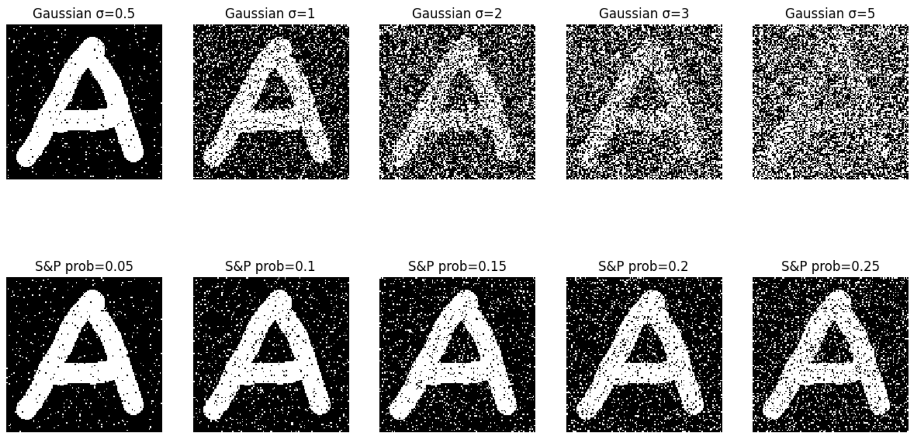
Q1 - Simulating Noisy Images

To evaluate the denoising capabilities of the ICM algorithm, we first simulate noisy versions of the original binary image using Gaussian and Salt-and-Pepper noise models.

Implementation

```python
def add_gaussian_noise(image, sigma):
    noise = np.random.normal(0, sigma, image.shape)
    noisy_image = image + noise
    noisy_image = np.where(noisy_image >= 0, 1, -1)
    return noisy_image

def add_salt_and_pepper_noise(image, prob):
    noisy_image = image.copy()
    rnd = np.random.rand(*image.shape)
    noisy_image[rnd < (prob / 2)] = 1
    noisy_image[rnd > 1 - (prob / 2)] = -1
    return noisy_image
```

Listing 2: Gaussian and Salt and Pepper Noise Functions

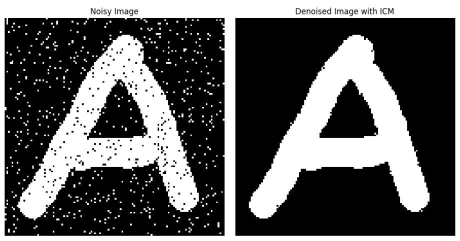


Q2 - Implementing the ICM Denoising Algorithm

The ICM algorithm was implemented to iteratively minimize the defined energy function and denoise the corrupted images. The algorithm's parameters were set as follows: $h = 0.5$, $\beta = 1.0$, and $\eta = 1.0$.

Implementation

```python
def icm_denoise(y, h, beta, eta, max_iter=50, tol=1e-5):
    x = y.copy()
    H, W = x.shape

    # Define the convolution kernel for 4-connected neighbors
    kernel = np.array([[0, 1, 0],
                       [1, 0, 1],
                       [0, 1, 0]])

    energies = []

    for it in range(max_iter):
        x_old = x.copy()

        # Compute the sum of neighbors for each pixel
        neighbors_sum = convolve2d(x, kernel, mode='same',
            boundary='wrap')

        # Compute the conditional energy for x_j = +1 and x_j = -1
        energy_plus = h * 1 - beta * neighbors_sum - eta * y
        energy_minus = h * (-1) + beta * neighbors_sum + eta * y

        # Update x based on the lower energy
        x = np.where(energy_plus < energy_minus, 1, -1)
```

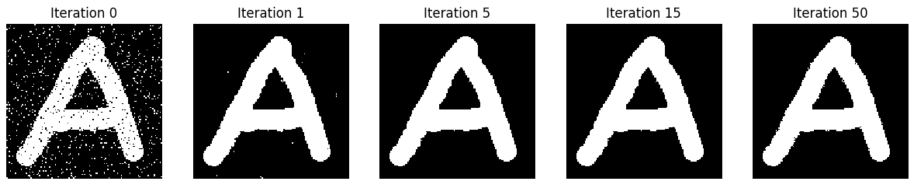


```python

        # Compute the total energy
        total_energy = h * np.sum(x) - beta * np.sum(x *
            neighbors_sum) - eta * np.sum(x * y)
        energies.append(total_energy)

        # Check for convergence
        changes = np.sum(x != x_old)
        if changes == 0:
            print("Convergence achieved.")
            break

    return x, energies
```

Listing 3: ICM Denoising Function

Q3 - Capturing Intermediate Snapshots During ICM Iterations

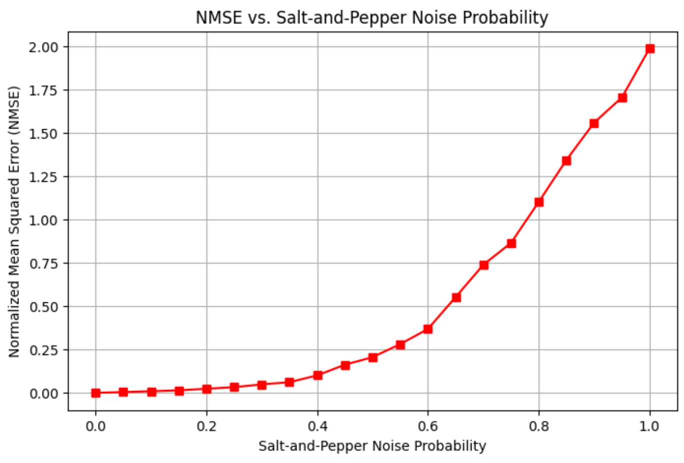
To visualize the denoising progression, snapshots of the denoised image were captured at specific iterations: 0 (initial noisy image), 1, 5, 15, and 50.



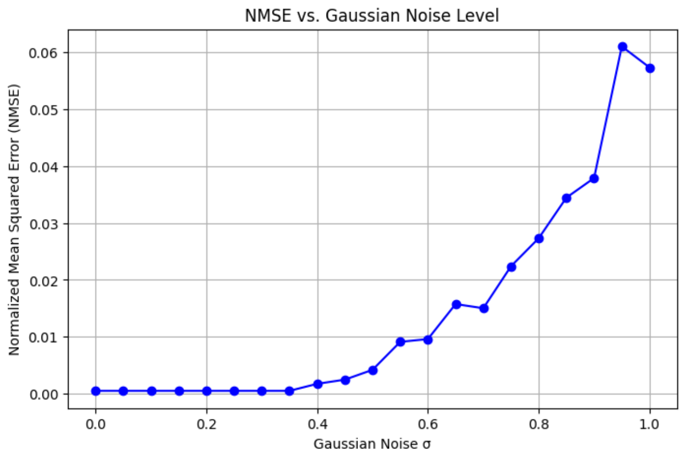
Q4 - Computing and Plotting NMSE for Various Noise Levels

Normalized Mean Squared Error (NMSE) was computed to quantify the denoising performance across different noise intensities.

```python
def compute_nmse(denoised, ground_truth):
    mse = np.mean((denoised - ground_truth) ** 2)
    norm = np.mean(ground_truth ** 2)
    nmse = mse / norm
    return nmse
```



Listing 4: NMSE Function



Q5 - Analysis of Denoising Performance with Increasing Noise Levels

The performance trends are visualized in the NMSE plots as functions of noise level, showing that NMSE increases nonlinearly with noise intensity.

- **Low Noise Levels:**
  - When the noise level is low ($\sigma \approx 0.1$ for Gaussian noise or $p \approx 0.05$ for Salt-and-Pepper noise), the ICM algorithm effectively denoises

the image.

  - Minimal changes are needed during the iterations, as the initial noisy image already provides a good approximation.

- **Moderate Noise Levels:**
  - For moderate noise levels ($\sigma \approx 0.5$ or $p \approx 0.2$), the algorithm performs well but requires more iterations to converge.
  - The NMSE increases slightly compared to low noise levels due to greater deviations between the noisy image and the ground truth.
  - Some fine details may be lost as the algorithm balances smoothness and data fidelity.

- **High Noise Levels:**
  - At high noise levels ($\sigma \geq 1.0$ or $p \geq 0.4$), the performance of the algorithm starts to degrade.
  - The algorithm struggles to recover the original image, leading to higher NMSE values.
  - Smoothness constraints (β) dominate the energy minimization, resulting in oversmoothed images where fine details and edges are lost.
  - Adherence to the noisy image (η) becomes counterproductive as the observed data is corrupted.

Q6 - Varying Parameters h, β, and η and Evaluating Their Impact

To understand the influence of each parameter on denoising performance, we modified the parameters by varying h, β, and η individually while keeping the other two parameters fixed.

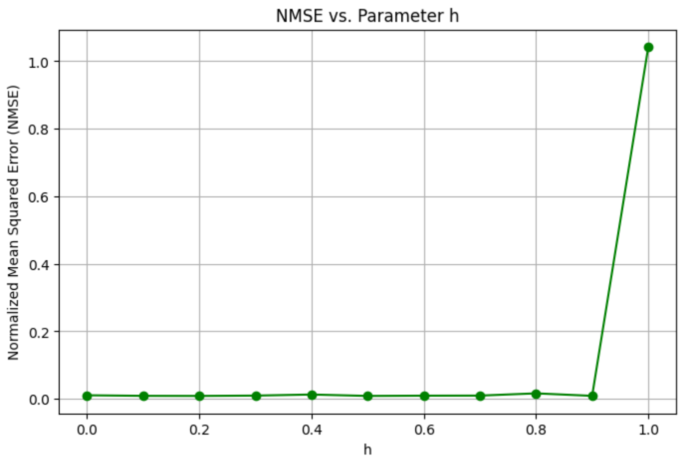
Implementation

```
    # Define parameter ranges
h_values = np.arange(0.0, 1.1, 0.1)
beta_values = np.arange(0.0, 2.2, 0.2)
eta_values = np.arange(0.0, 2.2, 0.2)

# Initialize dictionaries to store NMSE results
nmse_results = {
    'h': {},
    'beta': {},
    'eta': {}
}

# Select fixed values for other parameters during each sweep
fixed_params = {
    'h': 0.5,
    'beta': 1.0,
    'eta': 1.0
}


# Vary h, fix beta and eta

for h in h_values:
    current_h = h
    current_beta = fixed_params['beta']
    current_eta = fixed_params['eta']

    noisy_img = add_salt_and_pepper_noise(img, 0.10)
    denoised_img, energies = icm_denoise(noisy_img, current_h,
        current_beta, current_eta, max_iter=50)

    nmse = compute_nmse(denoised_img, img)
    nmse_results['h'][h] = nmse

    print(f"h={h:.1f}: NMSE={nmse:.4f}")


# Vary beta, fix h and eta

for beta in beta_values:
    current_h = fixed_params['h']
    current_beta = beta
    current_eta = fixed_params['eta']

    noisy_img = add_salt_and_pepper_noise(img, 0.10)
    denoised_img, energies = icm_denoise(noisy_img, current_h,
        current_beta, current_eta, max_iter=50)

    nmse = compute_nmse(denoised_img, img)
    nmse_results['beta'][beta] = nmse

    print(f"beta={beta:.1f}: NMSE={nmse:.4f}")


# Vary eta, fix h and beta

for eta in eta_values:
    current_h = fixed_params['h']
    current_beta = fixed_params['beta']
    current_eta = eta

    noisy_img = add_salt_and_pepper_noise(img, 0.10)
    denoised_img, energies = icm_denoise(noisy_img, current_h,
        current_beta, current_eta, max_iter=50)

    nmse = compute_nmse(denoised_img, img)
    nmse_results['eta'][eta] = nmse

    print(f"eta={eta:.1f}: NMSE={nmse:.4f}")
```

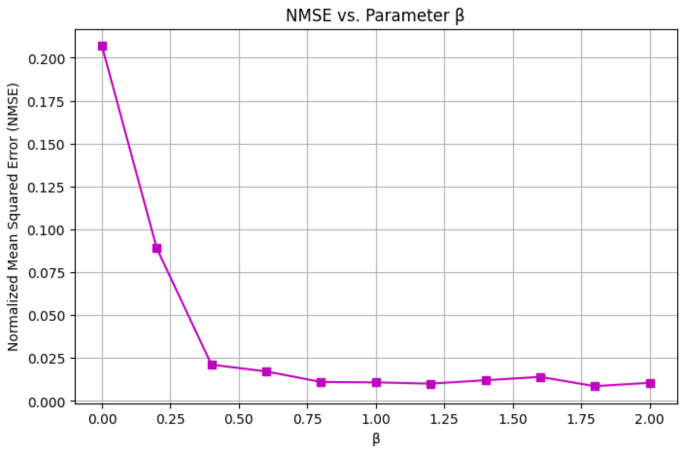
Q7 - Plotting NMSE Against Each Parameter

The following plots illustrate the relationship between each parameter and the NMSE, highlighting the optimal parameter settings that minimize NMSE.

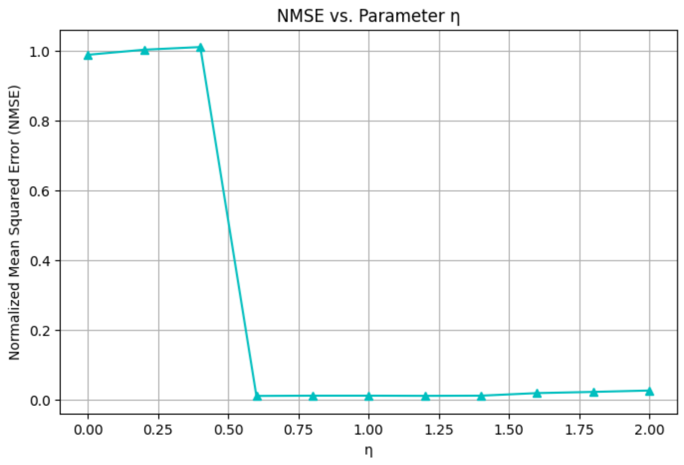
NMSE vs. Parameter h



NMSE vs. Parameter β



NMSE vs. Parameter β



Q8 - Discussion on Parameter Impacts

Varying h: Unary Term Coefficient

The parameter h introduces a bias towards assigning $+1$ or -1 to each pixel independently of its neighbors and the noisy data.

- **Low h Values:**

  Allow the algorithm to rely more on the pairwise (β) and data (η) terms. This facilitates effective denoising by leveraging neighborhood similarity. However, minimal bias may cause ambiguity in pixel states, especially in regions with conflicting information from neighbors.

- **High h Values:**

  Impose a strong bias towards a specific state, potentially leading to over-smoothing where the denoised image loses important details.

Varying β: Pairwise Term Coefficient

The parameter β governs the smoothness of the denoised image, neighboring pixels will adopt similar states.

- **Low β Values:**

  Reduce the emphasis on smoothness, allowing the denoised image to preserve edges and fine details. However, insufficient smoothness may result in residual noise being retained in the image.

- **High β Values:**

Improve smoothness by promoting uniformity among neighboring pixels. While this effectively removes noise, it risks oversmoothing important image features of the denoised image.

Varying η: Data Term Coefficient

The parameter η balances the adherence of the denoised image to the noisy observed data.

- **Low η Values:**

  Allow the algorithm to deviate more from the noisy data, facilitating greater noise correction. This can lead to lower NMSE as the denoised image aligns better with the ground truth. However, excessive deviation may distort genuine image features.

- **High η Values:**

  Enforce strong adherence to the noisy data, preventing the algorithm from effectively removing noise. This results in higher NMSE as residual noise persists in the denoised image.