

Image Reconstruction from Non-Uniform Samples

Using MM–CG with DCT– ℓ_p Prior

Project Report

M. Harsha Vardhan

SR No. 27110

M.Tech in Signal Processing

November 14, 2025

Contents

1	Derivation of the MM–CG Algorithm	2
1.1	Derivation of the MM–CG Algorithm (Expanded Explanation)	2
2	Experimental Setup	5
2.1	Test Images	5
2.2	Sampling Masks	5
2.3	Noise Model	5
2.4	Parameter Choices	5
3	Results and Discussion	6
3.1	Cameraman Image	6
3.1.1	Sampling Ratio $r = 0.1$	6
3.1.2	Sampling Ratio $r = 0.2$	7
3.1.3	Sampling Ratios $r = 0.3$ and $r = 0.5$	7
3.2	Lena Image	9
3.2.1	Sampling Ratios $r = 0.1$ and $r = 0.2$	9
3.2.2	Sampling Ratios $r = 0.3$ and $r = 0.5$	10
3.3	Tabular Data	11
4	Discussion	13
5	Conclusion	14
A	Python Implementation (MM–CG)	15

Chapter 1

Derivation of the MM–CG Algorithm

1.1 Derivation of the MM–CG Algorithm (Expanded Explanation)

In this work, we aim to reconstruct an unknown image $x^* \in \mathbb{R}^N$ from a random subset of its pixels. The observed measurements follow the model

$$m = Wx^* + \eta, \quad (1.1)$$

where W is a sampling operator that selects only M out of N pixels, and η is additive noise. We assume that x^* is sparse in the Discrete Cosine Transform (DCT) domain, that is, most coefficients of $y = \text{DCT}(x)$ are small or zero. Using this prior knowledge, we formulate the following regularized optimization problem:

$$\min_x J(x) = \|Wx - m\|_2^2 + \lambda \sum_{i=1}^N (\varepsilon + y_i^2)^p, \quad y = \text{DCT}(x), \quad (1.2)$$

where $0.2 < p < 0.5$ and $\varepsilon > 0$ ensures differentiability. The first term enforces consistency with the available samples, while the second promotes sparsity in the transform domain.

Why Majorization–Minimization?

The regularizer

$$R(y) = \sum_{i=1}^N (\varepsilon + y_i^2)^p$$

is smooth but *nonconvex*, and therefore minimizing $J(x)$ directly is computationally difficult and unstable. The MM framework solves such problems by iteratively constructing a simpler surrogate function that upper-bounds the original objective and is easier to minimize. Each MM step guarantees that the objective is non-increasing.

Because $(\varepsilon + t)^p$ is concave in t for $0 < p < 1$, we can majorize it using a first-order Taylor approximation:

$$(\varepsilon + y_i^2)^p \leq (\varepsilon + (y_i^{(k)})^2)^p + w_i^{(k)} \left(y_i^2 - (y_i^{(k)})^2 \right), \quad (1.3)$$

where

$$w_i^{(k)} = p(\varepsilon + (y_i^{(k)})^2)^{p-1}. \quad (1.4)$$

Dropping constants independent of x , we obtain the quadratic surrogate

$$R(y) \approx \sum_i w_i^{(k)} y_i^2. \quad (1.5)$$

Majorized Objective

Replacing the regularizer in (1.2), we obtain the surrogate objective

$$J_M(x) = \|Wx - m\|_2^2 + \lambda \sum_{i=1}^N w_i^{(k)} y_i^2. \quad (1.6)$$

This is now a *quadratic* function of x , and therefore much easier to minimize.

Using $y = \text{DCT}(x)$ and orthonormality of the DCT matrix D , we rewrite:

$$\sum_i w_i^{(k)} y_i^2 = \|\sqrt{w^{(k)}} \odot Dx\|_2^2. \quad (1.7)$$

Stationary Condition

To minimize $J_M(x)$, we set its gradient equal to zero:

$$\nabla_x J_M(x) = 2W^\top(Wx - m) + 2\lambda \text{IDCT}\left(w^{(k)} \odot \text{DCT}(x)\right) = 0. \quad (1.8)$$

Rearranging terms, we obtain the linear system

$$(W^\top W + \lambda \text{IDCT}\left(\text{diag}(w^{(k)}) \text{DCT}(\cdot)\right)) x = W^\top m. \quad (1.9)$$

Matrix-Free Operator Construction

Observe that $W^\top W$ is simply the binary sampling mask M , such that

$$(W^\top W)x = M \odot x,$$

and we define the operator

$$\mathcal{M}^{(k)}(z) = M \odot z + \lambda \text{IDCT}\left(w^{(k)} \odot \text{DCT}(z)\right). \quad (1.10)$$

Thus the MM update requires solving:

$$\mathcal{M}^{(k)}(x^{(k+1)}) = W^\top m. \quad (1.11)$$

Use of Conjugate Gradient

Since $N = 256^2 = 65536$ and storing $\mathcal{M}^{(k)}$ explicitly would require a dense $N \times N$ matrix with $\sim 4 \times 10^9$ entries, solving (1.9) by direct inversion is infeasible. However, CG only needs operator evaluations, not matrix storage.

Therefore we solve (1.9) using CG, initialized by $x^{(k)}$.

Final MM–CG Algorithm

1. Initialize $x^{(0)} = W^\top m$ (zero-filled reconstruction).
2. Repeat:
 - (a) $y^{(k)} = \text{DCT}(x^{(k)})$
 - (b) Compute weights $w_i^{(k)}$ from (1.4)
 - (c) Construct the linear operator $\mathcal{M}^{(k)}$
 - (d) Solve $\mathcal{M}^{(k)}(x) = W^\top m$ using CG
 - (e) $x^{(k+1)} = x$
3. Stop when $\frac{\|x^{(k+1)} - x^{(k)}\|_2}{\|x^{(k)}\|_2} < 10^{-4}$

This algorithm guarantees monotonic decrease of $J(x)$ and converges in practice within a few tens of MM iterations.

Chapter 2

Experimental Setup

2.1 Test Images

We use two standard 256×256 grayscale images:

- **Cameraman** (cameraman.tif),
- **Lena** (lena.png).

Images are converted to grayscale (if needed), resized to 256×256 , and normalized to $[0, 1]$.

2.2 Sampling Masks

For each image, we generate random binary masks for sampling ratios

$$r \in \{0.1, 0.2, 0.3, 0.5\}.$$

For a given r , exactly rN pixel locations are chosen uniformly at random and set to 1 in the mask; the remaining entries are 0.

2.3 Noise Model

We add white Gaussian noise to the sampled pixels to achieve an SNR of 30 dB. If $s = Wx^*$ denotes the ideal sampled signal, then the noise η is scaled to satisfy

$$\text{SNR} = 20 \log_{10} \frac{\|s\|_2}{\|\eta\|_2} \approx 30 \text{ dB}.$$

2.4 Parameter Choices

The parameters used in all experiments are:

- $\varepsilon = 10^{-6}$,
- $p \in \{0.3, 0.4, 0.5\}$,
- $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$,
- MM stopping tolerance: 10^{-4} ,
- CG tolerance: 10^{-6} .

For each combination of sampling ratio r , exponent p , and regularization λ , we reconstruct an image and compute PSNR and relative error. For each (image, r) pair, we report the best λ in terms of PSNR.

Chapter 3

Results and Discussion

3.1 Cameraman Image

3.1.1 Sampling Ratio $r = 0.1$

Figure 3.1 shows the sampling mask, noisy observation, and best reconstruction for the Cameraman image at sampling ratio $r = 0.1$. The corresponding PSNR vs. λ plot and convergence curve are shown in Figure 3.2.

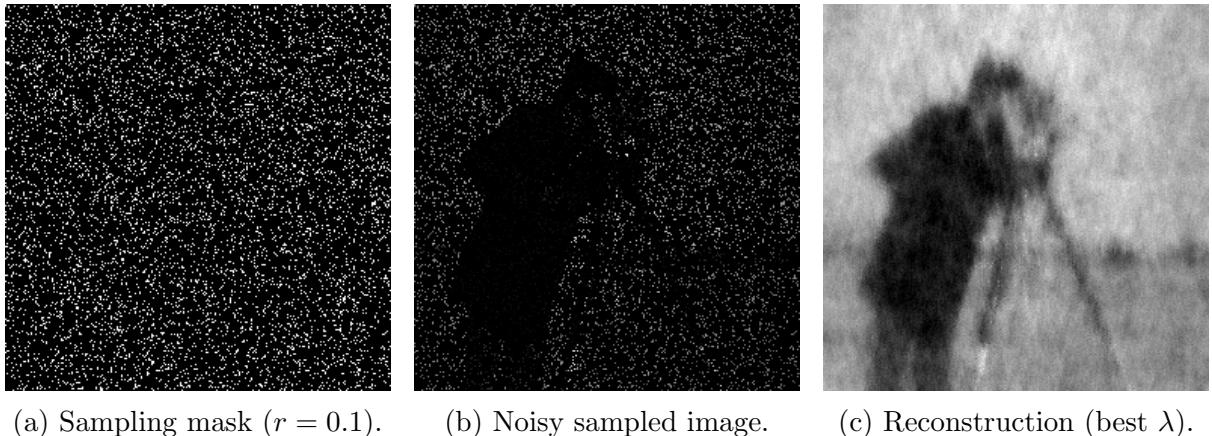


Figure 3.1: Cameraman, $r = 0.1$: mask, noisy observation, and reconstruction.

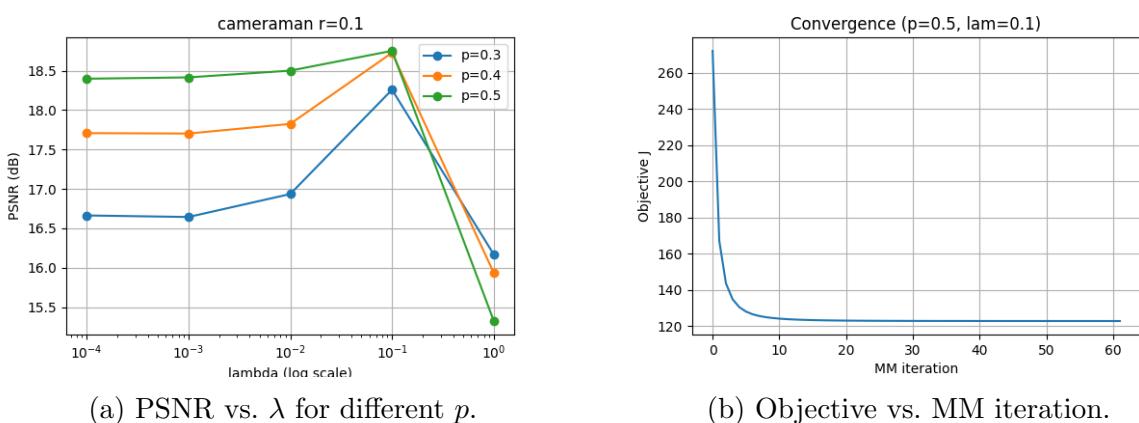


Figure 3.2: Cameraman, $r = 0.1$: PSNR and convergence curves.

3.1.2 Sampling Ratio $r = 0.2$

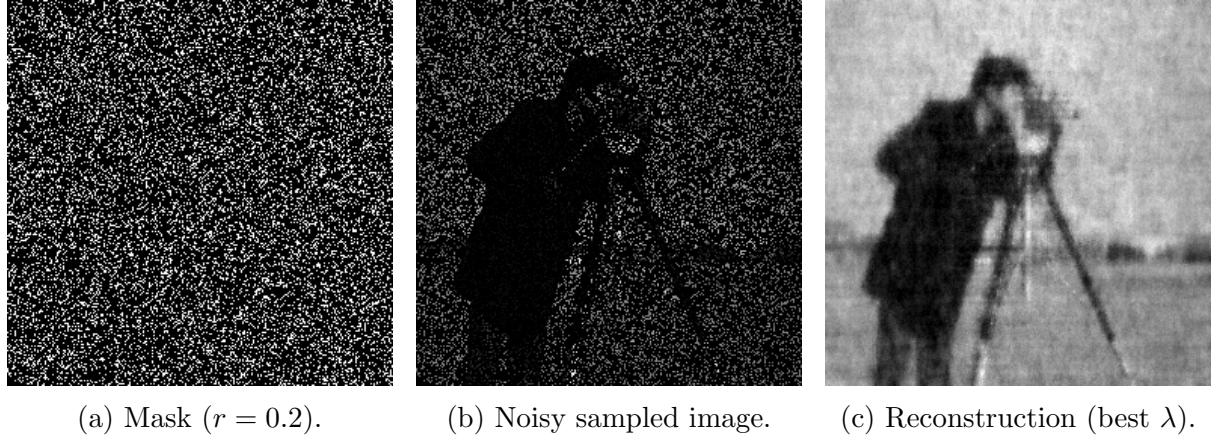


Figure 3.3: Cameraman, $r = 0.2$: mask, noisy observation, and reconstruction.

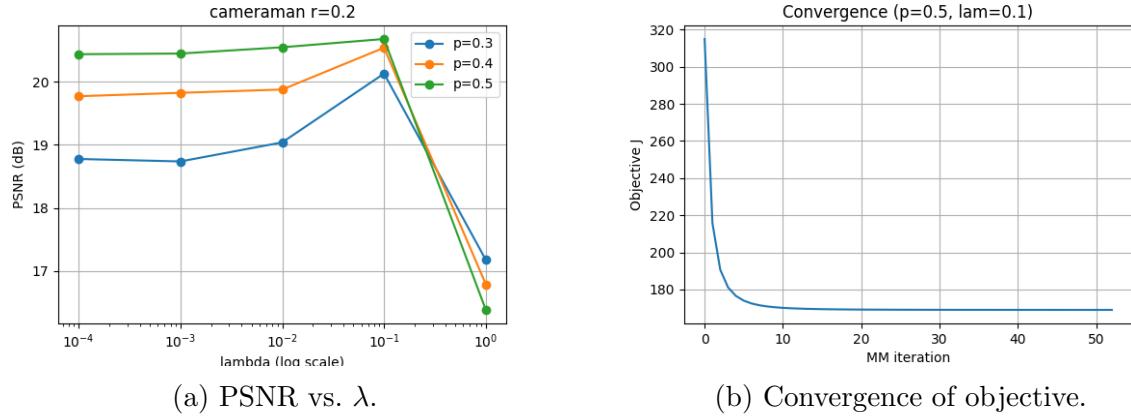


Figure 3.4: Cameraman, $r = 0.2$: PSNR and convergence.

3.1.3 Sampling Ratios $r = 0.3$ and $r = 0.5$

Similarly, Figures 3.5–3.8 show the results for $r = 0.3$ and $r = 0.5$.

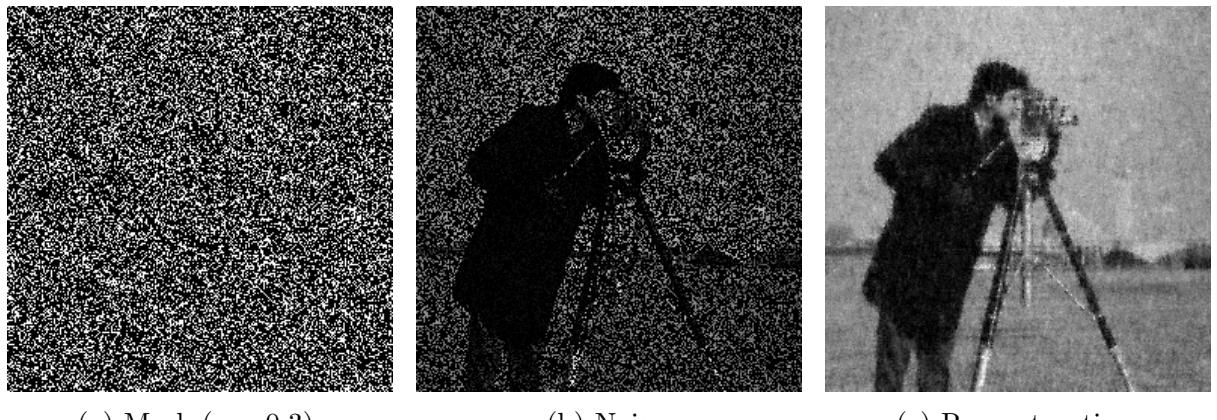


Figure 3.5: Cameraman, $r = 0.3$: mask, noisy, reconstruction.

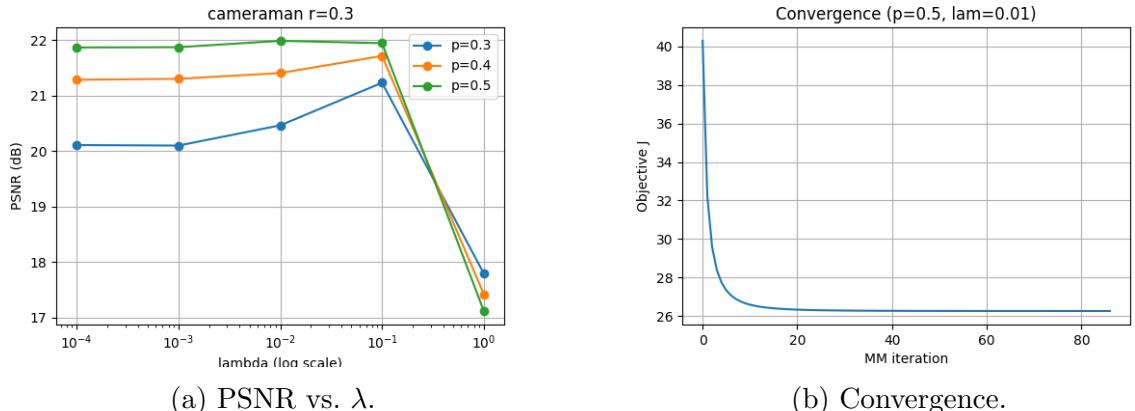


Figure 3.6: Cameraman, $r = 0.3$: PSNR and convergence.

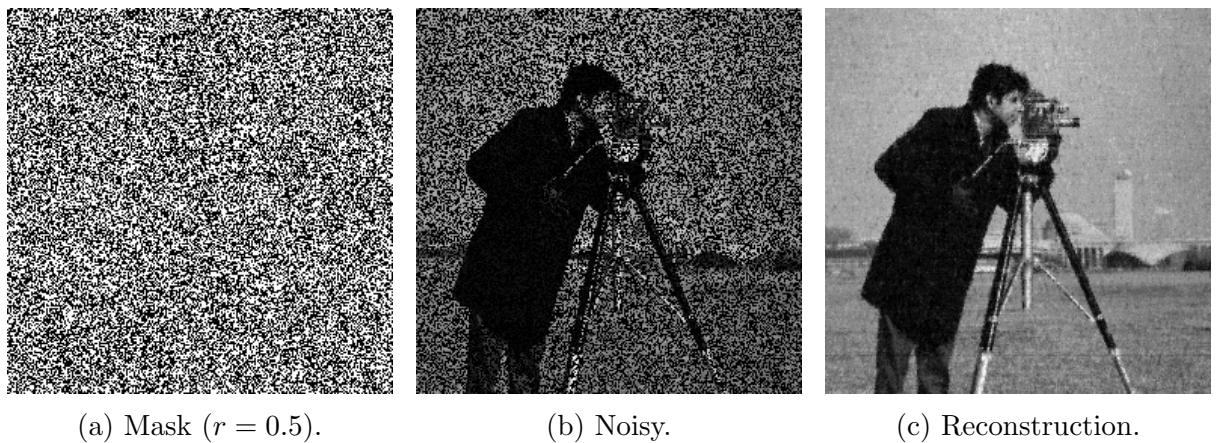


Figure 3.7: Cameraman, $r = 0.5$: mask, noisy, reconstruction.

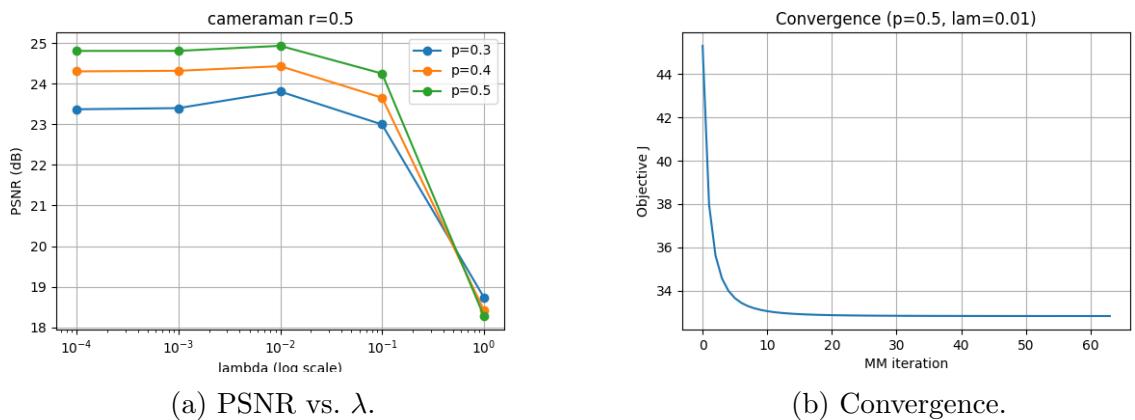


Figure 3.8: Cameraman, $r = 0.5$: PSNR and convergence.

3.2 Lena Image

3.2.1 Sampling Ratios $r = 0.1$ and $r = 0.2$

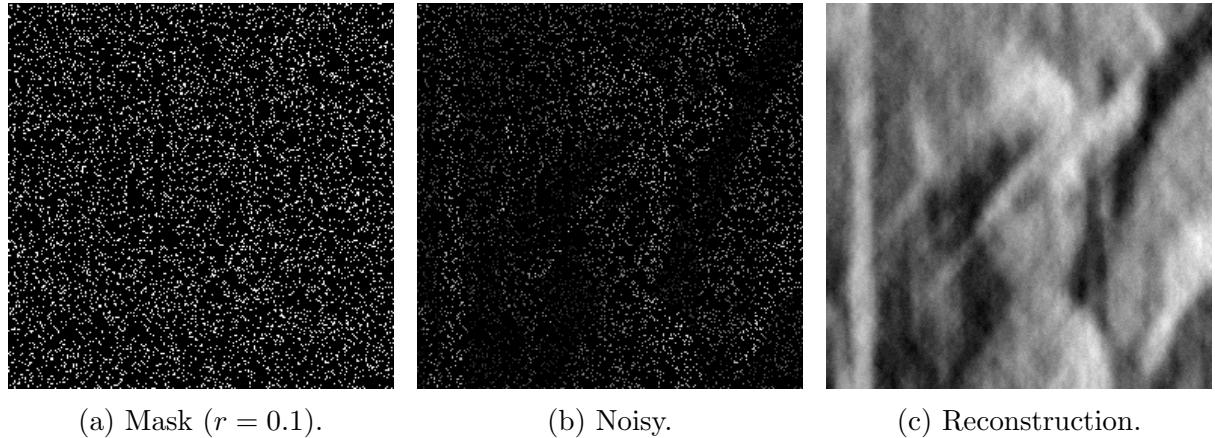


Figure 3.9: Lena, $r = 0.1$: mask, noisy, reconstruction.

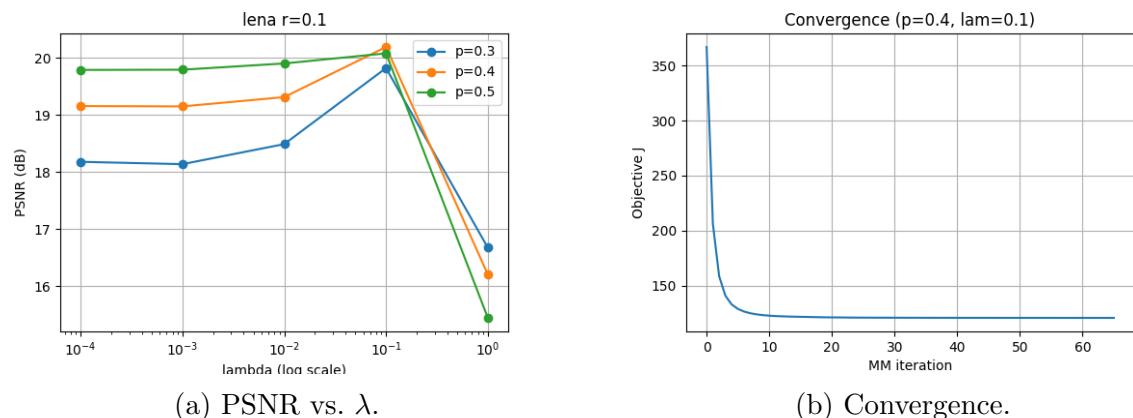


Figure 3.10: Lena, $r = 0.1$: PSNR and convergence.



Figure 3.11: Lena, $r = 0.2$: mask, noisy, reconstruction.

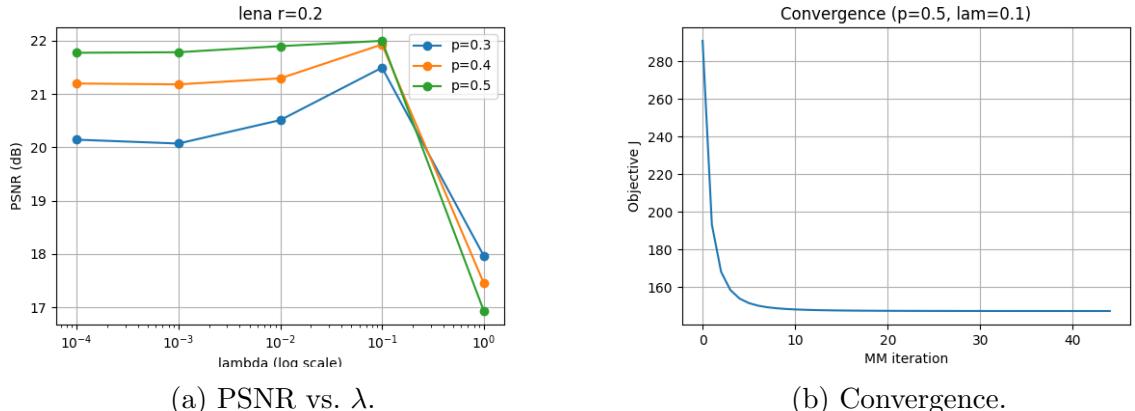


Figure 3.12: Lena, $r = 0.2$: PSNR and convergence.

3.2.2 Sampling Ratios $r = 0.3$ and $r = 0.5$



Figure 3.13: Lena, $r = 0.3$: mask, noisy, reconstruction.

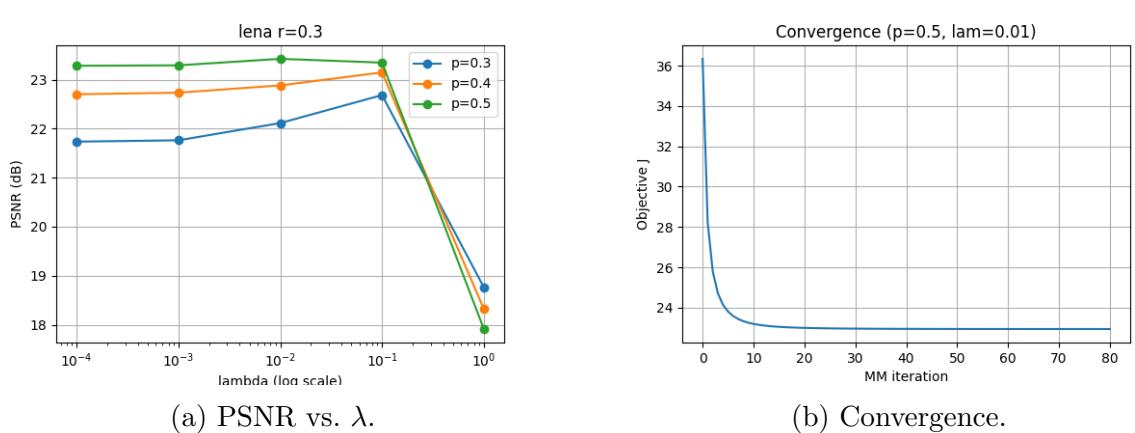


Figure 3.14: Lena, $r = 0.3$: PSNR and convergence.



(a) Mask ($r = 0.5$).

(b) Noisy.

(c) Reconstruction.

Figure 3.15: Lena, $r = 0.5$: mask, noisy, reconstruction.

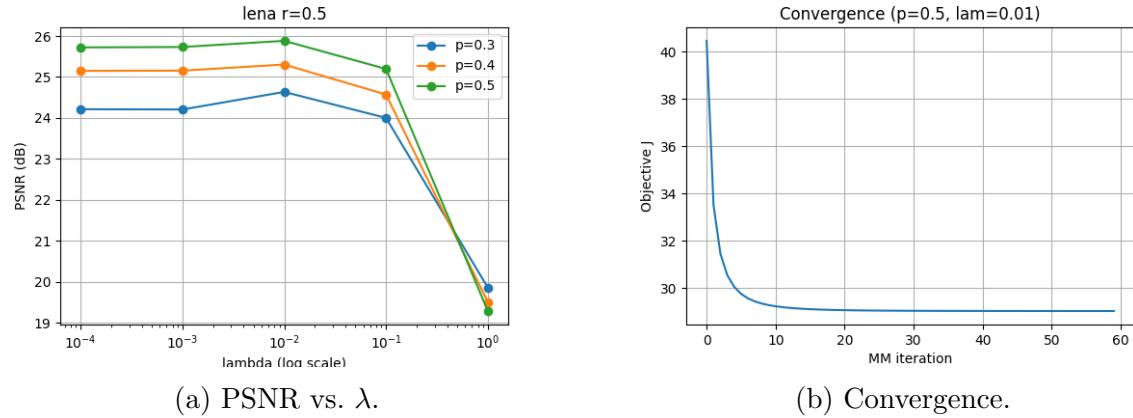


Figure 3.16: Lena, $r = 0.5$: PSNR and convergence.

3.3 Tabular Data

p	0.0001	0.001	0.01	0.1	1
0.3	16.66	16.64	16.93	18.26	16.17
0.4	17.71	17.70	17.82	18.73	15.93
0.5	18.40	18.41	18.50	18.75	15.32

Table 3.1: PSNR (dB) for Cameraman, $r = 0.1$, λ as columns

p	0.0001	0.001	0.01	0.1	1
0.3	18.78	18.74	19.04	20.13	17.17
0.4	19.77	19.83	19.88	20.54	16.79
0.5	20.44	20.45	20.55	20.68	16.38

Table 3.2: PSNR (dB) for Cameraman, $r = 0.2$

<i>p</i>	0.0001	0.001	0.01	0.1	1
0.3	20.11	20.10	20.46	21.23	17.79
0.4	21.28	21.30	21.40	21.71	17.42
0.5	21.87	21.87	21.99	21.94	17.12

Table 3.3: PSNR (dB) for Cameraman, $r = 0.3$

<i>p</i>	0.0001	0.001	0.01	0.1	1
0.3	23.37	23.40	23.80	22.99	18.74
0.4	24.30	24.32	24.43	23.65	18.43
0.5	24.80	24.81	24.93	24.25	18.28

Table 3.4: PSNR (dB) for Cameraman, $r = 0.5$

<i>p</i>	0.0001	0.001	0.01	0.1	1
0.3	18.18	18.14	18.49	19.82	16.67
0.4	19.15	19.15	19.31	20.19	16.20
0.5	19.79	19.79	19.90	20.08	15.44

Table 3.5: PSNR (dB) for Lena, $r = 0.1$, λ as columns

<i>p</i>	0.0001	0.001	0.01	0.1	1
0.3	20.14	20.07	20.51	21.49	17.96
0.4	21.19	21.18	21.29	21.93	17.45
0.5	21.77	21.78	21.89	21.99	16.93

Table 3.6: PSNR (dB) for Lena, $r = 0.2$

<i>p</i>	0.0001	0.001	0.01	0.1	1
0.3	21.73	21.76	22.11	22.68	18.76
0.4	22.70	22.73	22.88	23.15	18.32
0.5	23.28	23.29	23.42	23.35	17.91

Table 3.7: PSNR (dB) for Lena, $r = 0.3$

<i>p</i>	0.0001	0.001	0.01	0.1	1
0.3	24.21	24.21	24.63	24.00	19.86
0.4	25.15	25.15	25.30	24.57	19.50
0.5	25.72	25.73	25.88	25.20	19.28

Table 3.8: PSNR (dB) for Lena, $r = 0.5$

Chapter 4

Discussion

Overall, the experiments show that:

- **Effect of sampling ratio:** As the sampling ratio r increases, the reconstruction quality (PSNR) improves significantly for both images. At very low sampling ($r = 0.1$), the reconstructions are still recognizable but contain more artifacts and blurring. At $r = 0.5$, the reconstructions are visually very close to the ground truth.
- **Effect of p :** Smaller values of p (more nonconvex) tend to promote stronger sparsity in the DCT domain, often leading to sharper edges and better detail preservation. However, they can also make the optimization landscape more challenging and increase sensitivity to λ .
- **Effect of λ :** There is a clear trade-off in λ : too small a value under-regularizes and leaves noise/artifacts, whereas too large a value over-smooths the image. The PSNR vs. λ curves provide a clear way to select a good regularization level for each (r, p) combination.
- **Convergence:** The objective function is observed to decrease monotonically with MM iterations. The number of CG iterations per MM step remains reasonable, and using the previous iterate as initialization helps speed up convergence.

The method successfully reconstructs both the Cameraman and Lena images from as low as 10% random pixel samples with added noise, demonstrating the effectiveness of the MM–CG framework with a nonconvex DCT– ℓ_p prior.

Chapter 5

Conclusion

In this project, we formulated image reconstruction from non-uniform, noisy samples as a regularized optimization problem with a nonconvex DCT– ℓ_p prior. We derived and implemented a Majorization–Minimization algorithm with Conjugate Gradient inner loops, using only matrix-free operators (masking and DCT/IDCT transforms).

Experiments on standard test images (Cameraman and Lena) under different sampling ratios, noise levels, and hyperparameters showed that:

- The MM–CG scheme converges reliably and monotonically.
- The nonconvex prior with $0.3 \leq p \leq 0.5$ leads to high-quality reconstructions from heavily undersampled data.
- Proper tuning of λ is crucial and depends on both sampling ratio and choice of p .

This framework can be extended to other transform domains (e.g., wavelets) and more advanced measurement models, and forms a useful template for designing iterative algorithms for inverse problems with nonconvex priors.

Appendix A

Python Implementation (MM–CG)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct
from scipy.sparse.linalg import LinearOperator, cg
from PIL import Image
import time
import os

def dct2(x):
    # apply 1D dct (type II, orthonormal) on rows then cols
    return dct(dct(x.T, norm='ortho').T, norm='ortho')

def idct2(x):
    # inverse DCT (type III, orthonormal)
    return idct(idct(x.T, norm='ortho').T, norm='ortho')

def psnr(x_hat, x_true):
    N = x_true.size
    num = np.max(np.abs(x_true))
    den = np.linalg.norm((x_true - x_hat).ravel()) / np.sqrt(N)
    if den == 0:
        return np.inf
    return 20.0 * np.log10(num / den)

def rel_l2(x_hat, x_true):
    return np.linalg.norm(x_hat.ravel() - x_true.ravel())
    / np.linalg.norm(x_true.ravel())

def make_random_mask(shape, sampling_ratio, rng=None):
    rng = np.random.default_rng(rng)
    N = shape[0] * shape[1]
    M = int(np.round(sampling_ratio * N))
    idx = rng.choice(N, size=M, replace=False)
```

```

mask = np.zeros(N, dtype=np.uint8)
mask[idx] = 1
return mask.reshape(shape)

def apply_mask(x, mask):
    return mask * x

def mm_cg_reconstruct(m, mask, lam, p=0.4, eps=1e-6,
                      tol_cg=1e-6, tol_mm=1e-4, max_mm_iters=200,
                      verbose=False):
    shape = m.shape
    N = shape[0] * shape[1]
    # initialize zero-filled image
    xk = m.copy()
    Wt_m = m.copy() # because W^T m is just mask * m (measurements in place)
    diag_mask = mask # M operator is elementwise multiply by mask

    obj_vals = []
    rel_changes = []
    cg_iters_per_mm = []
    times = []

    for k in range(max_mm_iters):
        t0 = time.time()
        # DCT of current estimate
        y_hat = dct2(xk)
        # compute weights in DCT domain: w_i = p * (eps + y_i^2)^{p-1}
        w = p * (eps + y_hat**2)**(p - 1)

        # define linear operator M^{(k)} acting on image-shaped vectors
        def M_op(vec_flat):
            z = vec_flat.reshape(shape)
            term1 = diag_mask * z
            # DCT domain multiplication then IDCT
            Dz = idct2(w * dct2(z))
            return (term1 + lam * Dz).ravel()

        linop = LinearOperator((N, N), matvec=M_op, dtype=np.float64)

        # right-hand side is W^T m = mask * m
        b = Wt_m.ravel()

        # CG solve: M^{(k)} x = b
        # use previous xk as initial guess
        x0 = xk.ravel()
        x_sol, info = cg(linop, b, x0=x0, tol=tol_cg, maxiter=10*N)
        # info==0 : converged; >0 number of iterations, <0 error

```

```

x_next = x_sol.reshape(shape)

# diagnostics
# objective value J(x)
data_fidelity = np.linalg.norm(apply_mask(x_next, mask) - m)**2
y_next = dct2(x_next)
reg = lam * np.sum((eps + y_next**2)**p)
Jk = data_fidelity + reg
obj_vals.append(Jk)

rel_change = np.linalg.norm((x_next - xk)) / (np.linalg.norm(xk) + 1e-12)
rel_changes.append(rel_change)
cg_iters_per_mm.append(info if isinstance(info, (int, np.integer)) else 0)
times.append(time.time() - t0)

if verbose:
    print(f"MM iter {k:3d}: J={Jk:.6e}, rel_change={rel_change:.3e},"
          f"cg_info={info}")

# stopping
xk = x_next
if rel_change < tol_mm:
    if verbose:
        print("MM stopping criterion reached.")
    break

diagnostics = {
    'obj_vals': np.array(obj_vals),
    'rel_changes': np.array(rel_changes),
    'cg_iters': np.array(cg_iters_per_mm),
    'times': np.array(times),
    'iters': k + 1
}
return xk, diagnostics

def run_experiments(image_paths, sampling_ratios=[0.1,0.2,0.3,0.5],
                   p_list=[0.3,0.4,0.5],
                   lambda_grid=[1e-4,1e-3,1e-2,1e-1,1],
                   snr_db=30.0,
                   eps=1e-6,
                   rng_seed=0,
                   out_dir='results'):
    os.makedirs(out_dir, exist_ok=True)
    rng = np.random.default_rng(rng_seed)

    for img_path in image_paths:
        # load and normalize to [0,1]
        im = Image.open(img_path).convert('L')

```

```

im = im.resize((256,256))
x_true = np.asarray(im, dtype=np.float32) / 255.0

for r in sampling_ratios:
    mask = make_random_mask(x_true.shape, r, rng=rng)

    sampled = mask * x_true
    observed_energy = np.linalg.norm(sampled)**2
    eta_norm = np.linalg.norm(sampled) / (10** (snr_db / 20.0) + 1e-12)
    # create noise only at observed positions
    eta = np.zeros_like(x_true)

    noise_random = rng.standard_normal(size=x_true.shape)
    noise_random *= mask
    if np.linalg.norm(noise_random) == 0:
        noise_random = mask * 1e-12
    noise_random *= (eta_norm / np.linalg.norm(noise_random))
    eta = noise_random
    m = sampled + eta

# validation sweep: try all (p, lambda) combinations, keep best PSNR
best = {'psnr': -np.inf}
results_table = []
for p in p_list:
    for lam in lambda_grid:
        x_hat, diag = mm_cg_reconstruct(m, mask, lam, p=p, eps=eps,
                                         tol_cg=1e-6, tol_mm=1e-4, max_mm_-
        cur_psnr = psnr(x_hat, x_true)
        cur_rel = rel_l2(x_hat, x_true)
        results_table.append((p, lam, cur_psnr, cur_rel, diag))
        if cur_psnr > best['psnr']:
            best = {'p': p, 'lam': lam, 'psnr': cur_psnr, 'rel': cur_rel,}

# save best results and summary plots
base_name = os.path.splitext(os.path.basename(img_path))[0]
out_prefix = f"{base_name}_r{int(r*100)}"

# image outputs
plt.imsave(os.path.join(out_dir, f"{out_prefix}_mask.png"),
mask, cmap='gray')
plt.imsave(os.path.join(out_dir, f"{out_prefix}_noisy.png"),
np.clip(m, 0, 1), cmap='gray')
plt.imsave(os.path.join(out_dir, f"{out_prefix}_recon_p{best['p']}_
_lam{best['lam']}.png"),
np.clip(best['x_hat'], 0, 1), cmap='gray')

# PSNR vs lambda plot (for each p)
plt.figure(figsize=(6,4))
for p in p_list:

```

```

lambdas = []
psnrs = []
for (pp, lam, ps, rl, dg) in results_table:
    if pp == p:
        lambdas.append(lam)
        psnrs.append(ps)
lambdas = np.array(lambdas)
psnrs = np.array(psnrs)
# sort by lambda for log plot
order = np.argsort(lambdas)
plt.plot(lambdas[order], psnrs[order], marker='o', label=f"p={p}")
plt.xscale('log')
plt.xlabel('lambda (log scale)')
plt.ylabel('PSNR (dB)')
plt.title(f"{base_name} r={r}")
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(out_dir, f"{out_prefix}_psnr_vs_lambda.png"))
plt.close()

# save convergence curves for the best setting
diag = best['diag']
plt.figure(figsize=(6,4))
plt.plot(diag['obj_vals'])
plt.xlabel('MM iteration')
plt.ylabel('Objective J')
plt.title(f"Convergence (p={best['p']}, lam={best['lam']})")
plt.grid(True)
plt.savefig(os.path.join(out_dir, f"{out_prefix}_convergence.png"))
plt.close()

# print summary to console
print(f"{base_name} r={r}: best p={best['p']}, lam={best['lam']}, PSNR={best['psnr']:.2f} dB, rel={best['rel']:.4f}")

print("All experiments finished. Results saved in:", out_dir)

if __name__ == "__main__":
    image_paths = ["cameraman.tif", "lena.png"]
    run_experiments(image_paths,
                    sampling_ratios=[0.1,0.2,0.3,0.5],
                    p_list=[0.3,0.4,0.5],
                    lambda_grid=[1e-4,1e-3,1e-2,1e-1,1],
                    snr_db=30.0,
                    eps=1e-6,
                    rng_seed=42,
                    out_dir='results_mm_cg')

```

