ECE 592 Homework 4

Kudiyar Orazymbetov (korazym@ncsu.edu)
Nico Casale (ncasale@ncsu.edu)

November 15, 2017

Note that the entries are links.

Contents

List of Figures

Listings

1 Wavelet Experiment

1.1 Comparison of methods

The distortion value for the different wavelet-based compression methods is given in the table below

Compression Alg.	Distortion
ezw	25.681
spiht	59.991
stw	50.453
wdr	25.681
aswdr	25.681
spiht_3d	314.79
lvl_mmc	76.358
gbl_mmc_f	3380.8
gbl_mmc_h	3380.8

Table 1: Distortion values of different wavelet-based compression algorithms.

The distortion value from our kmeans implementation for $P = 2$, $R = 1$ is 50.22. The 'ezw', 'wdr', and 'aswdr' give the best, *i.e.* lowest distortion values; all being equal to 25.681.

1.2 Embedded Zerotree Wavelet (EZW) image compression

In the embedded zerotree wavelet image compression (EZW), the discrete wavelet transform is accomplished using hierarchical sub-bands. Sub-bands will be logarithmically spaced. They are populated by applying filters to isolate each sub-band's frequencies from the original signal. Then, an image will be decomposed into wavelets. A zerotree (data structure) is used to map the the most significant wavelets. These efficiently encode the image. Zerotrees improve the compression of the original signal. A wavelet coefficient is regarded to be insignificant when $|x| < T$, where T is some threshold. The scanning of coefficients is performed starting from the parent; the child coefficient is insignificant if parent coefficient is insignificant. Then, a sequence of thresholds will be applied to determine the significance. The coding is accomplished in an embedded way, which means that there will be a sequence of binary decisions that enable the identification of an image from the null-image. The total cost of this approach is the sum of the cost of significance map and cost of non-zero values. It has 2 advantages over kmeans. EZW can precisely control the encoding rate, and no pre-training is required. It uses a discrete wavelet transform which is also a good to capture the important features, rather than clusters that do not carry much detail. Please see the final section for our code implementation of this experiment.

2 AMP Implementation

2.1 Introduction

In this experiment, we implemented approximate message passing (AMP) for an N-length Rademacher signal, x. A Rademacher signal takes the following values with pmf:

$$f(x) = \begin{cases} \theta & if\ x = -1 \\ 1 - 2\theta & if\ x = 0 \\ \theta & if\ x = 1 \end{cases} \tag{2.1}$$

Essentially, 2θ determines the expected number of non-zeros in the signal. For instance, if $N = 2000, \theta = 0.05$, we can expect there to be roughly 200 non-zero entries in x. Pictured below is a stem plot that illustrates a sample of the Rademacher signal.

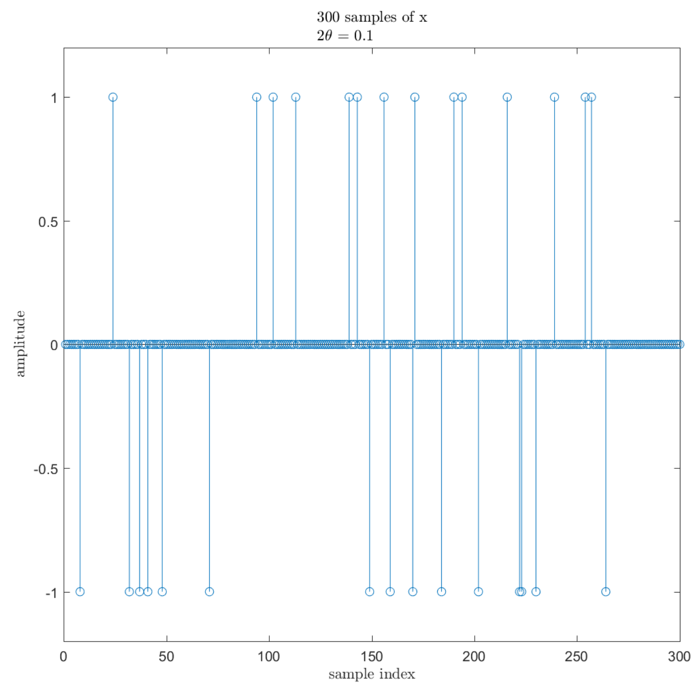

Figure 2.1: 300 samples of the Rademacher signal.

To run AMP, we generate a measurement matrix, $A \in \mathbb{R}^{m \times n}$, where $M = round(\delta * N)$. δ is the measurement rate. In general, higher measurement rates yield a better reconstruction. We show the results of an experiment where we varied δ in section 2.3.2. Note that the elements of A are $N(0, \sqrt{\frac{1}{M}})$, which yields unit-norm columns on average.

In addition, we need to generate a noise signal that disrupts the ability of AMP to reconstruct the signal. Given a signal-to-noise ratio (SNR) of γ in dB, we generate noise $z \sim N(0, \sqrt{\gamma}) \in \mathbb{R}^M$. After generating the noise and measurement matrices, we can form the noisy observations, $y = Ax + z \in \mathbb{R}^M$.

2.2 AMP Implementation and Results

To complete the AMP implementation, we started with code written by Dr. Baron from the course web-page. The main modification to the script was in `denoise.m`, where we utilized a conditional denoiser instead of the original Weiner filter based denoiser (which is specific to denoising a Bernoulli Gaussian random variable.) The conditional expectation denoiser evaluates $E[x|v]$ as

$$\begin{aligned} E[x|v] &= (-1) * Pr(x=-1|v) + (0) * Pr(x=0|v) + (1) * Pr(x=1|v) \\ &= Pr(x=1|v) - Pr(x=-1|v) \end{aligned} \tag{2.2}$$

Where $Pr(x=1|v)$ is given by Bayes' Rule as

$$\begin{aligned} Pr(x=1|v) &= \frac{Pr(x=1|v) \cdot f(v)}{f(v)} \\ &= \frac{f(x=1,v)}{f(v)} \\ &= \frac{Pr(x=1) \cdot f(V=v|x=1)}{f(x=-1,v) + f(x=0,v) + f(x=1,v)} \\ &= \frac{\theta \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v-1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v+1)^2}{2\sigma^2}} + \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-v^2}{2\sigma^2}} + \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v-1)^2}{2\sigma^2}}} \end{aligned} \tag{2.3}$$

Note that $Pr(x=-1|v)$ is defined similarly, albeit with $Pr(x=-1) \cdot f(V=v|x=-1) = \theta \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v+1)^2}{2\sigma^2}}$ in the numerator. Taking these quantities, $E[x|v]$ is given by

$$\begin{aligned} E[x|v] &= Pr(x=1|v) - Pr(x=-1|v) \\ &= \frac{f(x=1,v) - f(x=-1,v)}{f(v)} \\ &= \frac{Pr(x=1) \cdot f(V=v|x=1) - Pr(x=-1) \cdot f(V=v|x=-1)}{f(x=-1,v) + f(x=0,v) + f(x=1,v)} \\ &= \frac{\theta \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v-1)^2}{2\sigma^2}} - \theta \cdot \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v+1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v+1)^2}{2\sigma^2}} + \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-v^2}{2\sigma^2}} + \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v-1)^2}{2\sigma^2}}} \end{aligned} \tag{2.4}$$

Where σ is the variance of the Gaussian noise in z, $\frac{1}{\sqrt{\gamma}}$.

Note that the denoiser also needs to estimate the derivative of $E[x|v]$. We do so by adding a perturbation value of $1e-10$ to each element of v. Then we compute $E[x|v]$ using the perturbed v and calculate the piece-wise derivative as $\frac{\hat{x}-\hat{x}_{perturbed}}{perturbation}$.

2.2.1 AMP Results

After initializing the meta-parameters described earlier, we generate all the values and run AMP. Below is an example of our reconstruction on some samples of the input signal.

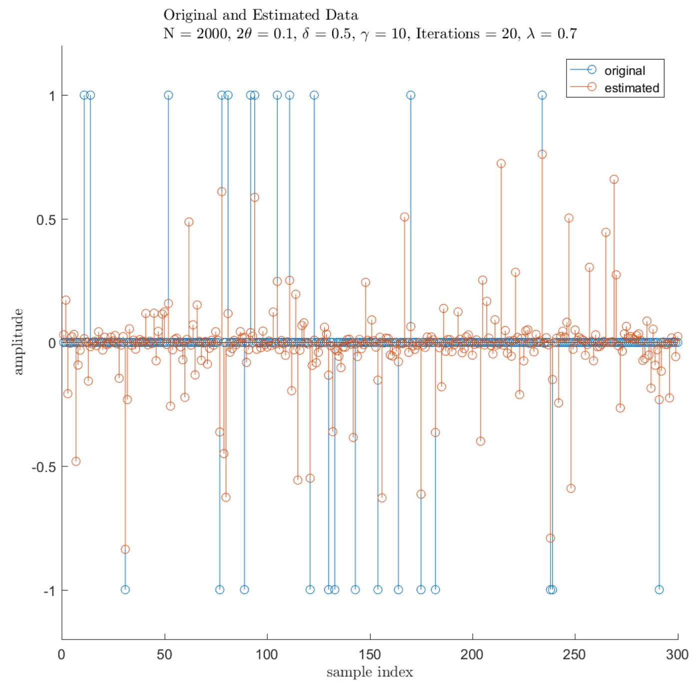

Figure 2.2: Result of running AMP on a input signal of length $N = 2000$.

In addition, the mean-squared error (MSE) at each iteration of the AMP algorithm is pictured below. We ran all of our simulations with 20 iterations of AMP. Note that λ is a damping parameter that slows the update step of the estimate of x, $\hat{x}$. As a reminder, 2θ is the probability of non-zeros in x, δ is the measurement rate, which is proportional to the size of M, and γ is the SNR in dB, which dictates the variance (and thereby magnitude) of z.

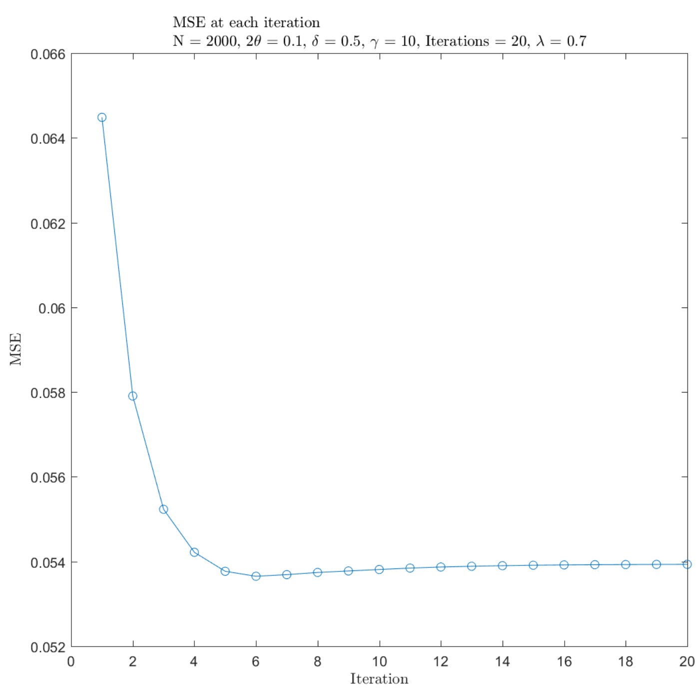

Figure 2.3: MSE at each iteration of an execution of AMP.

2.3 Experiments in Varying Parameters γ, δ, and N

2.3.1 Effects of Varying γ

For this experiment, we varied the value of γ, which is the SNR of Ax with respect to the noise z. We find that higher SNRs yield a better reconstruction, as evidenced in the figure below. We obtained this plot by holding all parameters fixed except γ. Note that the title says 'Minimum' MSE as we took only the minimum value of the MSE across the 20 iterations. On occasion, the MSE actually increases with iterations after the initial drop. This is due to variations in the noise and measurement matrices.This phenomena can be observed in fig. (2.3).

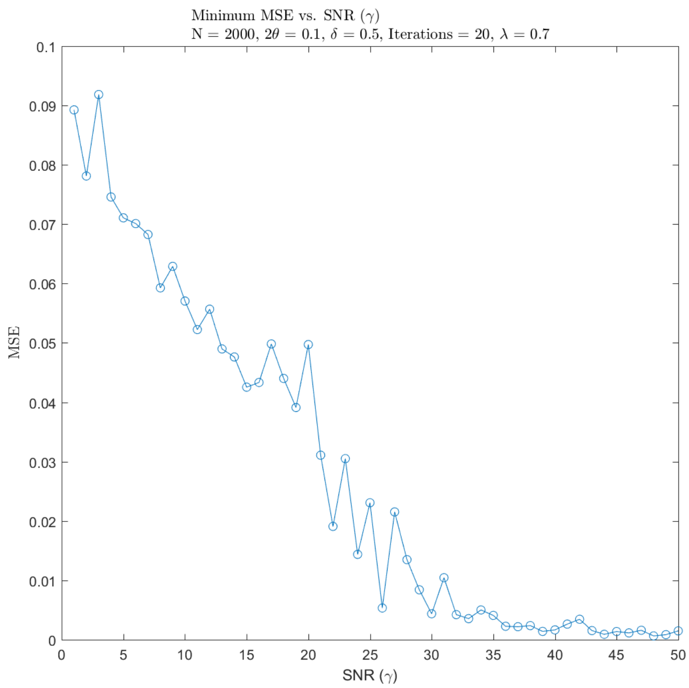

Figure 2.4: Effects of varying γ on the MSE of AMP.

2.3.2 Effects of Varying δ

For this experiment, we varied the value of δ, which is the measurement rate of A. It determines the size in the first dimension of A, M. We find that higher measurement rates yield a better reconstruction, as evidenced in the figure below. We obtained this plot by holding all parameters fixed except δ.

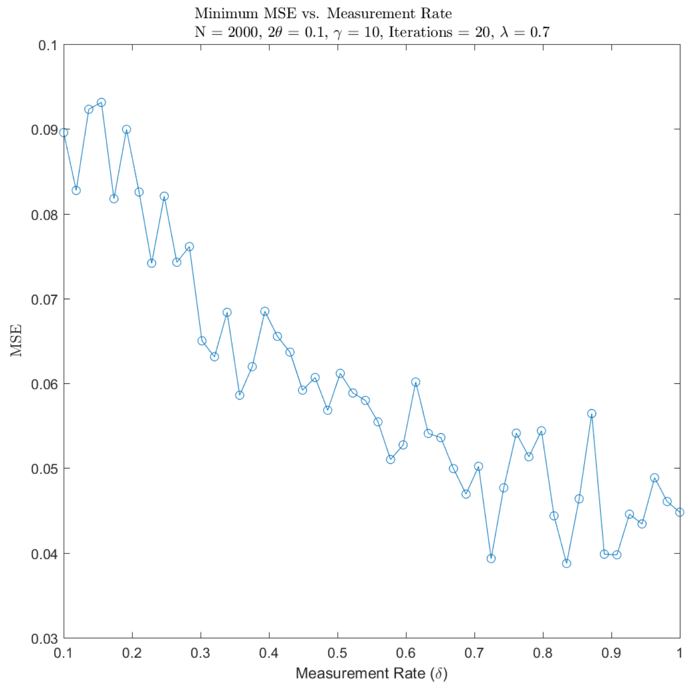

Figure 2.5: Effects of varying δ on the MSE of AMP.

2.3.3 Effects of Varying N

For this experiment, we varied the value of N, which is the number of elements in x. It determines the size in the second dimension of A, N. We find that higher numbers of original measurements yield a better reconstruction, as evidenced in the figure below. We obtained this plot by holding all parameters fixed except N. Note that these values were obtained by running AMP on each value of N 5 times. The average of these 5 experiments was used in the plot. This was an attempt to smooth out the plot, but as you can see it wasn't very successful. In this way, notice that the variance of the MSE between values of N tends to decrease as N increases. We infer that larger values of N introduce more stability in the reconstruction across initialization. These larger values of N are more robust to less appropriate initial values of A and z.

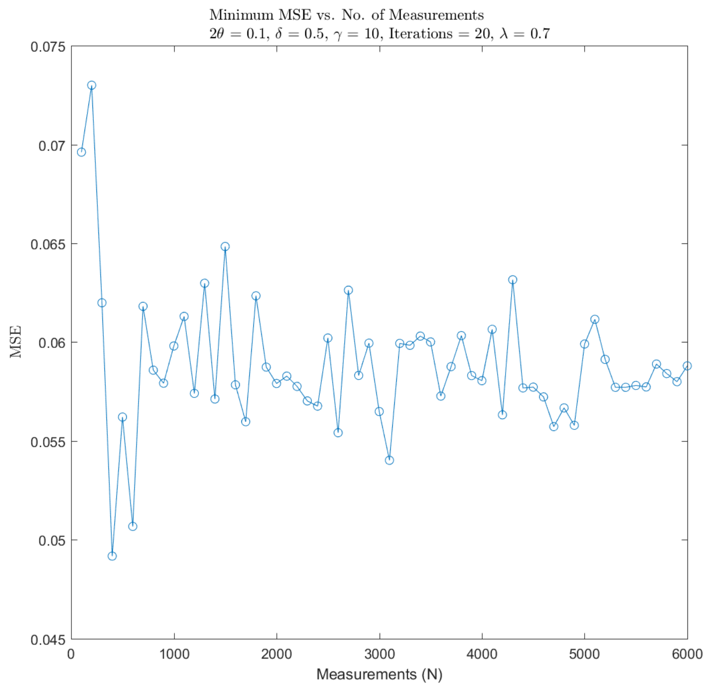

Figure 2.6: Effects of varying N on the MSE of AMP.

2.4 Comparison of AMP and *ell-1* Recovery (Linear Programming)

As a final experiment, we compared the accuracy and time to execute of our AMP implementation and an implementation of ell-1 recovery. The latter is a linear programming method that utilizes the built-in MATLAB function `linprog(.)`. Following the example of the source code on the course website, we implemented a linear programming wrapper function that sets up the parameters to `linprog(.)` and returns the reconstructed signal and MSE. We captured 5 executions of AMP and ell-1 recovery for values of $N \in \{100, 200, ..., 700\}$. Below is a plot of the MSE and execution time for these experiments. Note that the time plot for AMP is at the very bottom.

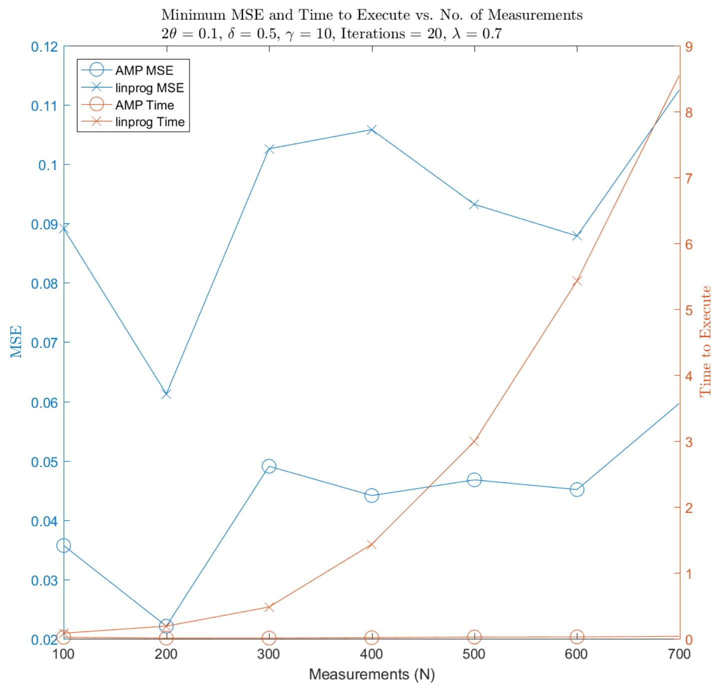

Figure 2.7: A comparison in MSE and Time between ell-1 and AMP reconstructions.

We find that AMP seems to have a much better runtime compared to the linear programming implementation. The linear programming is seemingly quadratic in runtime, while the AMP seems more linear. In addition, AMP yields consistently lower MSE values than linear programming.

3 Code Listings

3.1 Main Code for Problem 1

Listing 1: Code to calculate distortion with different wavelet-based methods.

```
%{
ECE592
Hw4 Problem 1
 Nico Casale
 Kudiyar Orazymbetov
%}

%%
%clear; close all;
addpath('utility', '../images');
setup();
global imagesFolder
overwriteImage = 0;

fprintf('ECE 592 HW 4\n');
fprintf(strcat(datestr(now),'\n'));
% read image
I = double(imread('image3.gif'));
[M, N] = size(I);
%%
P = 2; % square patch dimension
R = 0.25:0.25:1; % various values for Rate
K = round(2 .^(R*P^2));
for j = 1:length(K)
   % partition into patches
   Ipartitioned = im2col(I, [P P], 'distinct');

   % apply k-means
   [idx, Cn] = kmeans(Ipartitioned', K(j));

   % reconstruct image
   indexrepresentations = zeros(length(idx), P^2);
   for i = 1:length(idx)
      indexrepresentations(i, :) = Cn(idx(i), :);
   end
   Iquantized = col2im(indexrepresentations', [P P], size(I), 'distinct');

   subplot(1, 2, 1);
   imshow(I, []);
   f2 = subplot(1,2,2);
   imshow(Iquantized, []);

   % Calculate distortion D
   D(j) = sum(sum((I - Iquantized) .^2)/(M*N));
end
%%
%methods = {'ezw', 'spiht'; 'stw'; 'wdr'; 'aswdr'; 'spiht_3d'; 'lvl_mmc'; 'gbl_mmc_f'; 'gbl_mmc_h'};
methods = {'ezw', 'spiht', 'stw', 'wdr', 'aswdr', 'spiht_3d', 'lvl_mmc', 'gbl_mmc_f', 'gbl_mmc_h'};
```

```
%methods = ['ezw', 'spiht', 'stw', 'wdr', 'aswdr', 'spiht_3d', 'lvl_mmc', 'gbl_mmc_f', 'gbl_mmc_h
    '];

%methods = cellstr(methods);
mse = zeros(9,1);
for i=1:9
    if i<7
        [cr,bpp] = wcompress('c',I,'I.wtc',methods{i},'maxloop',12);
    elseif i == 7
        [cr,bpp] = wcompress('c',I,'I.wtc',methods{i},'maxloop',12);
    elseif i > 7
        [cr,bpp] = wcompress('c',I,'I.wtc',methods{i},'maxloop',12);
    end
    Xc = wcompress('u','I.wtc');
    delete('I.wtc')
    D1 = abs(I-Xc).^2;
    mse(i) = sum(D1(:))/numel(I);
end
%%
T = array2table(mse, 'RowNames', methods)
```

3.2 Main Code for Problem 2

Listing 2: Code to solve Problem 2.

```
%{
ECE 592 hw4 problem 2

n casale
ncasale@ncsu.edu

Kudiyar Orazymbetov
korazym@ncsu.edu

AMP simulation
17/11/9
%}

% initialize
clear; close all;
addpath('utility', '..');

global imagesFolder
imagesFolder = '../images/';
addpath(imagesFolder);
overwriteImages = 1;

set(0,'defaultTextInterpreter','latex');

fprintf(strcat('ECE 592 HW4 P2\n', datestr(now),'\n'));

% initial meta-parameters
% x
N = 2000;
theta = 0.05; % (1/2)*pr(non-zero)
% A
```

```
delta = 0.5; % measurement rate
% z
gamma = 10; % SNR, dB
% AMP
iterations = 20;
lambda = 0.7; % damping

%% AMP execution

[x, xhat, mse, ~, ~] = AMP(1, N, theta, delta, gamma, iterations, lambda);

figure(1);
if (0)
   file = sprintf('rademacher');
   file = strcat(imagesFolder, file);
   print(file, '-dpng');
end

figure(2);
if (0)
   file = sprintf('AMP');
   file = strcat(imagesFolder, file);
   print(file, '-dpng');
end

figure(3);
if (0)
   file = sprintf('AMP_mse');
   file = strcat(imagesFolder, file);
   print(file, '-dpng');
end

%% part f

% i varied SNR (gamma)
gammas = 1:50;
mses = zeros(1,length(gammas));

for gamma = gammas

   [x, xhat, mse, ~, ~] = AMP(0, N, theta, delta, gamma, iterations, lambda);
   mses(gamma) = min(mse);

end

% plot results
f = instantiateFig(4);
plot(gammas, mses, 'o-');
title(sprintf('Minimum MSE vs. SNR ($$\\gamma$$)\nN = %d, $$2\\theta$$ = %0.1f, $$\\delta$$ = %0.1f, Iterations = %d, $$\\lambda$$ = %0.1f', ...
   N, 2*theta, delta, iterations, lambda));
xlabel('SNR (\gamma)', 'Interpreter', 'tex')
ylabel('MSE')
prettyPictureFig(f);

if (0)
```

```
    file = sprintf('vary_gamma');
    file = strcat(imagesFolder, file);
    print(file, '-dpng');
end

%% ii varied measurement rate (delta)

gamma = 10; % SNR, dB, re-init from i
deltas = linspace(0.1, 1, 50);

for delta = deltas

    [x, xhat, mse, ~, ~] = AMP(0, N, theta, delta, gamma, iterations, lambda);
    mses(find(deltas == delta)) = min(mse);

end

% plot results
f = instantiateFig(5);
plot(deltas, mses, 'o-');
title(sprintf('Minimum MSE vs. Measurement Rate\nN = %d, $$2\\theta$$ = %0.1f, $$\\gamma$$ = %d, Iterations = %d, $$\\lambda$$ = %0.1f', ...
    N, 2*theta, gamma, iterations, lambda));
xlabel('Measurement Rate (\delta)', 'Interpreter', 'tex')
ylabel('MSE')
prettyPictureFig(f);

if (0)
    file = sprintf('vary_delta');
    file = strcat(imagesFolder, file);
    print(file, '-dpng');
end

%% iii varied N

delta = 0.5; % re-init from ii
Ns = 100:100:6000;
mses = zeros(1, length(Ns));
REPS = 5;

for N = Ns
    fprintf('     N = %d\n',N);
    for rep = 1:REPS
        [x, xhat, mse, ~, ~] = AMP(0, N, theta, delta, gamma, iterations, lambda);
        mses(find(Ns == N)) = mses(find(Ns == N)) + min(mse);
    end
end

mses = mses./REPS;

% plot results
f = instantiateFig(6);
plot(Ns, mses, 'o-');
title(sprintf('Minimum MSE vs. No. of Measurements\n$$2\\theta$$ = %0.1f, $$\\delta$$ = %0.1f, $$\\gamma$$ = %d, Iterations = %d, $$\\lambda$$ = %0.1f', ...
    2*theta, delta, gamma, iterations, lambda));
```

```matlab
xlabel('Measurements (N)', 'Interpreter', 'tex')
ylabel('MSE')
prettyPictureFig(f);

if (0)
   file = sprintf('vary_n');
   file = strcat(imagesFolder, file);
   print(file, '-dpng');
end

%% linear programming comparison with AMP

% initial meta-parameters
% x
theta = 0.05;
% A
delta = 0.5; % measurement rate
% z
gamma = 10; % SNR, dB
% AMP
iterations = 20;
lambda = 0.7; % damping

% compare speed and MSE
REPS = 5;
Ns = 100:100:700;
times = zeros(2, length(Ns));
mses = ones(2, length(Ns))*inf;

for N = Ns
   fprintf('     N = %d\n',N);
   for rep = 1:REPS
      tic;
      [x, xhat, mse, A, y] = AMP(0, N, theta, delta, gamma, iterations, lambda);
      times(1, find(Ns == N)) = times(1,find(Ns == N)) + toc;
      mses(1, find(Ns == N)) = min(min(mse), mses(1, find(Ns == N)));

      tic;
      [xhat, mse] = sim_linprog(N, A, x, y);
      times(2, find(Ns == N)) = times(2,find(Ns == N)) + toc;
      mses(2, find(Ns == N)) = min(min(mse), mses(2, find(Ns == N)));
   end
end

times = times./5; % get avg time

% then plot MSE vs. N
f = instantiateFig(7);
yyaxis left
plot(Ns, mses(1,:), 'o-', Ns, mses(2,:), 'x-', 'MarkerSize', 10);
ylabel('MSE')
title(sprintf('Minimum MSE and Time to Execute vs. No. of Measurements\n$$2\\theta$$ = %0.1f, $$\\delta$$ = %0.1f, $$\\gamma$$ = %d, Iterations = %d, $$\\lambda$$ = %0.1f', ...
   2*theta, delta, gamma, iterations, lambda));
yyaxis right
plot(Ns, times(1,:), 'o-', Ns, times(2,:), 'x-', 'MarkerSize', 10);
```

```
ylabel('Time to Execute')
xlabel('Measurements (N)', 'Interpreter', 'tex')
legend('AMP MSE', 'linprog MSE', 'AMP Time', 'linprog Time', 'Location', 'northwest');
prettyPictureFig(f);

if (0)
   file = sprintf('ell1');
   file = strcat(imagesFolder, file);
   print(file, '-dpng');
end
```

3.3 Supporting Functions for Problem 2

Listing 3: AMP implementation.

```
%{
ECE 592 hw4 problem 2

n casale
ncasale@ncsu.edu

Kudiyar Orazymbetov
korazym@ncsu.edu

AMP simulation
17/11/9
%}

function [x, xhat, mse, A, y] = AMP(plot_bool, N, theta, delta, gamma, iterations, lambda)

   % generate signal
   var_x = 2*theta;
   mask = binornd(1,2*theta, [N,1]);
   x = (rand(N,1)<.5)*2 - 1;
   x = mask.*x;

   % plot a subset of the signal
   if plot_bool
      f = instantiateFig(1);
      samps = 300;
      stem(x(1:samps));
      prettyPictureFig(f);
      ylim([-1.2 1.2]);
      title(sprintf('%d samples of x\n$$2\\theta$$ = %0.1f', samps, 2*theta));
      xlabel('sample index');
      ylabel('amplitude');
   end

   % measurement matrix A in R^{M x N}
   M = round(N*delta);
   sigma = sqrt(1/M);
   A = sigma.*randn(M,N);
   AT = A';

   % noise
   z = sqrt(1/gamma).*randn(M,1);
```

```

% noisy measurements
y = A*x + z;

% Dr. Baron's AMP implementation
% initialization
mse = zeros(iterations,1); % store mean square error
xhat = zeros(N,1); % estimate of signal
dt = zeros(N,1); % derivative of denoiser
rt = zeros(M,1); % residual

for iter = 1:iterations

    % update residual
    rt = y - A*xhat + 1/delta*mean(dt)*rt;
    % compute pseudo-data
    vt = xhat + AT*rt;
    % estimate scalar channel noise variance; estimator is due to Montanari
    var_t = mean(rt.^2);
    % denoising
    [xt1, dt] = denoise(vt, var_x, var_t, theta);
    % damping step
    xhat = lambda*xt1 + (1-lambda)*xhat;
    mse(iter) = mean((xhat-x).^2);

end

% plot estimated values over original
if plot_bool
   f = instantiateFig(2); hold on;
   stem(x(1:samps));
   stem(xhat(1:samps));
   ylim([-1.2 1.2]);
   hold off;
   legend('original', 'estimated');
   title(sprintf('Original and Estimated Data\nN = %d, $$2\\theta$$ = %0.1f, $$\\delta$$ =
       %0.1f, $$\\gamma$$ = %d, Iterations = %d, $$\\lambda$$ = %0.1f', ...
      N, theta, delta, gamma, iterations, lambda));
   xlabel('sample index');
   ylabel('amplitude');
   prettyPictureFig(f);
end

% tanaka's fixed point equation
n = linspace(0.01, 1, iterations);
mmse = ((1./n) - 1).*(delta/gamma);

% plot mse
if plot_bool
   f = instantiateFig(3); %hold on;
   plot(mse,'o-');
   %plot(mmse,'x-');
   %hold off;
   title(sprintf('MSE at each iteration\nN = %d, $$2\\theta$$ = %0.1f, $$\\delta$$ = %0.1f, $$
       \\gamma$$ = %d, Iterations = %d, $$\\lambda$$ = %0.1f', ...
      N, 2*theta, delta, gamma, iterations, lambda));
```

```
        xlabel('Iteration')
        ylabel('MSE')
        prettyPictureFig(f);
    end

    fprintf('AMP error = %10.6f\n', min(mse));

end
```

Listing 4: AMP Denoising Function.

```
%{
ECE 592 hw4 problem 2

n casale
ncasale@ncsu.edu

Kudiyar Orazymbetov
korazym@ncsu.edu

AMP simulation
Denoising function
utilized Dr. Baron's code from the course webpage
17/11/9
%}

function [xhat, d] = denoise(v, var_x, var_z, epsilon)

    term1 = (1 - 2*epsilon)*normpdf(v, 0, sqrt(var_z));
    term2 = epsilon*normpdf(v, 1, sqrt(var_x + var_z));
    term3 = epsilon*normpdf(v, -1, sqrt(var_x + var_z));
    xhat = (term2 - term3)./(term1 + term2 + term3); % denoised version, x(t+1)

    % empirical derivative
    Delta = 1e-10; % perturbation
    term1_d = (1 - 2*epsilon)*normpdf(v + Delta, 0, sqrt(var_z));
    term2_d = epsilon*normpdf(v + Delta, 1, sqrt(var_x + var_z));
    term3_d = epsilon*normpdf(v + Delta, -1,sqrt(var_x + var_z));
    xhat2 = (term2_d - term3_d)./(term1_d + term2_d + term3_d);
    d = (xhat2 - xhat)/Delta;

end
```

Listing 5: Linear Programming Implementation.

```
%{
ECE 592 hw4 problem 2

n casale
ncasale@ncsu.edu

Kudiyar Orazymbetov
korazym@ncsu.edu

linear programming ell1 solver
used Dr. Baron's example code from the course webpage
17/11/9
```

```
%}

function [xell1, mse] = sim_linprog(N, A, x, y)

   %-----------
   % linear programming
   %-----------
   % xhat = argmin ||x||_1 s.t. y=A*x
   % take x=xpos-xneg
   % xpos>=0, xneg>=0
   % xhat=argmin xpos+xneg s.t. y=A*(xpos-xneg)
   fprintf('solving ell1\n')
   f=ones(2*N,1);
   Aeq=[A -A];
   beq=y;
   lb=zeros(2*N,1); % lower bound zero
   ub=ones(2*N,1)*inf; % upper bound is infinity
   xsolve=linprog(f,[],[],Aeq,beq,lb,ub);
   xp=xsolve(1:N);
   xn=xsolve(N+1:2*N);
   xell1=xp-xn;
   mse = mean((xell1-x).^2);
   fprintf('ell1 error = %10.6f\n', mse);

end
```