Table of Contents


```
clc;
clear;
tic;
```

Algorithm Description

```
% This algorithm denoises RGB images by performing gradient descent
 and
% minimizing the loss function on each channel separately.
```

Reading images

```
groundTruth = imread("../data/histology_noiseless.png");
noisyImg = imread("../data/histology_noisy.png");
[r, c, n] = size(noisyImg);

groundTruth = im2double(groundTruth);
noisyImg = im2double(noisyImg);

channel1 = noisyImg(:, :, 1);
channel2 = noisyImg(:, :, 2);
channel3 = noisyImg(:, :, 3);
```

Error calculation

```
oldRRMSE = RRMSE(groundTruth, noisyImg);

fprintf("Original errors\n------------------\n");
fprintf("RRMSE between noiseless and low noise images = %f\n",
 oldRRMSE);
```

```
Original errors
------------------
RRMSE between noiseless and low noise images = 0.199383
```

Denoising using quadratic loss

```
% hyperparams per channel
```

```
alpha1 = 0.7;
alpha2 = 0.8;
alpha3 = 0.6;

[denoisedImgChan1, loss11] = gradientDescent(channel1, alpha1, 0, 1);
[denoisedImgChan2, loss12] = gradientDescent(channel2, alpha2, 0, 1);
[denoisedImgChan3, loss13] = gradientDescent(channel3, alpha3, 0, 1);

denoisedImg1 = cat(3, denoisedImgChan1, denoisedImgChan2,
 denoisedImgChan3);

newRRMSE = RRMSE(groundTruth, denoisedImg1);

fprintf("Optimal alphas per channel = [ %f %f %f ]\n", alpha1, alpha2,
 alpha3);

fprintf("Quadratic error\n-----------------\n");

fprintf("RRMSE between noiseless and denoised img :- \n");
fprintf("At alpha = %f\n", newRRMSE);
[temp1, ~] = gradientDescent(channel1, 1.2 * alpha1, 0, 1);
[temp2, ~] = gradientDescent(channel2, 1.2 * alpha2, 0, 1);
[temp3, ~] = gradientDescent(channel3, 1.2 * alpha3, 0, 1);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At 1.2 * alpha = %f\n", error);
[temp1, ~] = gradientDescent(channel1, 0.8 * alpha1, 0, 1);
[temp2, ~] = gradientDescent(channel2, 0.8 * alpha2, 0, 1);
[temp3, ~] = gradientDescent(channel3, 0.8 * alpha3, 0, 1);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At 0.8 * alpha = %f\n\n", error);
```

```
Optimal alphas per channel = [ 0.700000 0.800000 0.600000 ]
Quadratic error
------------------
RRMSE between noiseless and denoised img :-
At alpha = 0.057671
At 1.2 * alpha = 0.074995
At 0.8 * alpha = 0.062057
```

Gradient descent using Huber loss

```
% hyperparams per channel
alpha1 = 0.8;
alpha2 = 0.8;
alpha3 = 0.9;
gamma1 = 0.12;
gamma2 = 0.2;
gamma3 = 0.25;
```

```
[denoisedImgChan1, loss21] = gradientDescent(channel1, alpha1, gamma1,
 2);
[denoisedImgChan2, loss22] = gradientDescent(channel2, alpha2, gamma2,
 2);
[denoisedImgChan3, loss23] = gradientDescent(channel3, alpha3, gamma3,
 2);

denoisedImg2 = cat(3, denoisedImgChan1, denoisedImgChan2,
 denoisedImgChan3);

newRRMSE = RRMSE(groundTruth, denoisedImg2);

fprintf("Optimal alphas per channel = [ %f %f %f ]\n", alpha1, alpha2,
 alpha3);
fprintf("Optimal gammas per channel = [ %f %f %f ]\n", gamma1, gamma2,
 gamma3);

fprintf("Huber error\n------------------\n");

fprintf("RRMSE between noiseless and denoised img :- \n");
fprintf("At alpha, gamma = %f\n", newRRMSE);
[temp1, ~] = gradientDescent(channel1, 1.2 * alpha1, gamma1, 2);
[temp2, ~] = gradientDescent(channel2, 1.2 * alpha2, gamma2, 2);
[temp3, ~] = gradientDescent(channel3, 1.2 * alpha3, gamma3, 2);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At 1.2 * alpha, gamma = %f\n", error);
[temp1, ~] = gradientDescent(channel1, 0.8 * alpha1, gamma1, 2);
[temp2, ~] = gradientDescent(channel2, 0.8 * alpha2, gamma2, 2);
[temp3, ~] = gradientDescent(channel3, 0.8 * alpha3, gamma3, 2);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At 0.8 * alpha, gamma = %f\n", error);
[temp1, ~] = gradientDescent(channel1, alpha1, 1.2 * gamma1, 2);
[temp2, ~] = gradientDescent(channel2, alpha2, 1.2 * gamma2, 2);
[temp3, ~] = gradientDescent(channel3, alpha3, 1.2 * gamma3, 2);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At alpha, 1.2 * gamma = %f\n", error);
[temp1, ~] = gradientDescent(channel1, alpha1, 0.8 * gamma1, 2);
[temp2, ~] = gradientDescent(channel2, alpha2, 0.8 * gamma2, 2);
[temp3, ~] = gradientDescent(channel3, alpha3, 0.8 * gamma3, 2);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At alpha, 0.8 * gamma = %f\n\n", error);
```

```
Optimal alphas per channel = [ 0.800000 0.800000 0.900000 ]
Optimal gammas per channel = [ 0.120000 0.200000 0.250000 ]
Huber error
------------------
RRMSE between noiseless and denoised img :-
At alpha, gamma = 0.052677
At 1.2 * alpha, gamma = Inf
At 0.8 * alpha, gamma = 0.064563
```

```
At alpha, 1.2 * gamma = 0.052940
At alpha, 0.8 * gamma = 0.053084
```

Gradient descent using custom DAF

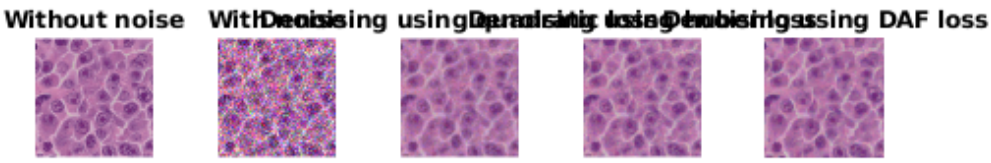
```
% hyperparams per channel
alpha1 = 0.8;
alpha2 = 0.8;
alpha3 = 0.9;
gamma1 = 0.12;
gamma2 = 0.2;
gamma3 = 0.25;

[denoisedImgChan1, loss31] = gradientDescent(channel1, alpha1, gamma1,
 3);
[denoisedImgChan2, loss32] = gradientDescent(channel2, alpha2, gamma2,
 3);
[denoisedImgChan3, loss33] = gradientDescent(channel3, alpha3, gamma3,
 3);

denoisedImg3 = cat(3, denoisedImgChan1, denoisedImgChan2,
 denoisedImgChan3);

newRRMSE = RRMSE(groundTruth, denoisedImg3);

fprintf("Optimal alphas per channel = [ %f %f %f ]\n", alpha1, alpha2,
 alpha3);
fprintf("Optimal gammas per channel = [ %f %f %f ]\n", gamma1, gamma2,
 gamma3);

fprintf("DAF error\n------------------\n");

fprintf("RRMSE between noiseless and denoised img :- \n");
fprintf("At alpha, gamma = %f\n", newRRMSE);
[temp1, ~] = gradientDescent(channel1, 1.2 * alpha1, gamma1, 3);
[temp2, ~] = gradientDescent(channel2, 1.2 * alpha2, gamma2, 3);
[temp3, ~] = gradientDescent(channel3, 1.2 * alpha3, gamma3, 3);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At 1.2 * alpha, gamma = %f\n", error);
[temp1, ~] = gradientDescent(channel1, 0.8 * alpha1, gamma1, 3);
[temp2, ~] = gradientDescent(channel2, 0.8 * alpha2, gamma2, 3);
[temp3, ~] = gradientDescent(channel3, 0.8 * alpha3, gamma3, 3);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At 0.8 * alpha, gamma = %f\n", error);
[temp1, ~] = gradientDescent(channel1, alpha1, 1.2 * gamma1, 3);
[temp2, ~] = gradientDescent(channel2, alpha2, 1.2 * gamma2, 3);
[temp3, ~] = gradientDescent(channel3, alpha3, 1.2 * gamma3, 3);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At alpha, 1.2 * gamma = %f\n", error);
```

```
[temp1, ~] = gradientDescent(channel1, alpha1, 0.8 * gamma1, 3);
[temp2, ~] = gradientDescent(channel2, alpha2, 0.8 * gamma2, 3);
[temp3, ~] = gradientDescent(channel3, alpha3, 0.8 * gamma3, 3);
temp = cat(3, temp1, temp2, temp3);
error = RRMSE(groundTruth, temp);
fprintf("At alpha, 0.8 * gamma = %f\n\n", error);
```

```
Optimal alphas per channel = [ 0.800000 0.800000 0.900000 ]
Optimal gammas per channel = [ 0.120000 0.200000 0.250000 ]
DAF error
------------------
RRMSE between noiseless and denoised img :-
At alpha, gamma = 0.049392
At 1.2 * alpha, gamma = 0.073896
At 0.8 * alpha, gamma = 0.057438
At alpha, 1.2 * gamma = 0.049335
At alpha, 0.8 * gamma = 0.050685
```

Results

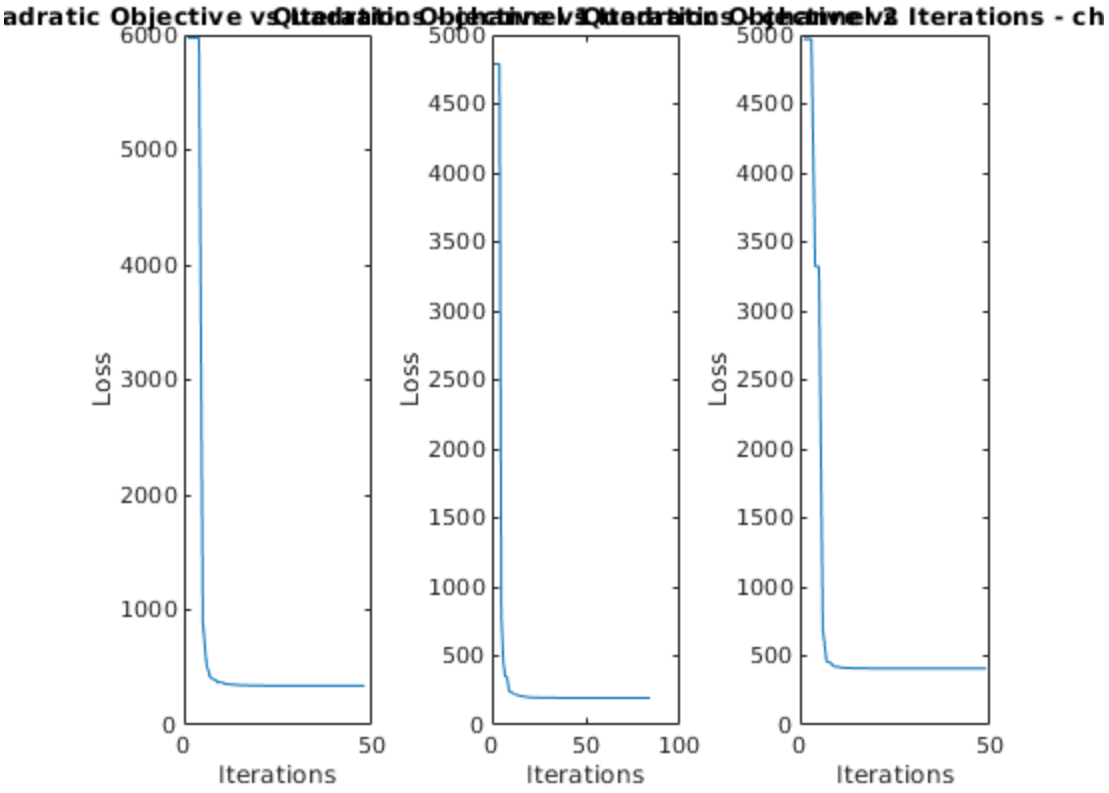
```
figure;

subplot(1, 5, 1);
imshow(uint8(255 * groundTruth));
title("Without noise");
subplot(1, 5, 2);
imshow(uint8(255 * noisyImg));
title("With noise");
subplot(1, 5, 3);
imshow(uint8(255 * denoisedImg1));
title("Denoising using quadratic loss");
subplot(1, 5, 4);
imshow(uint8(255 * denoisedImg2));
title("Denoising using huber loss");
subplot(1, 5, 5);
imshow(uint8(255 * denoisedImg3));
title("Denoising using DAF loss");
```

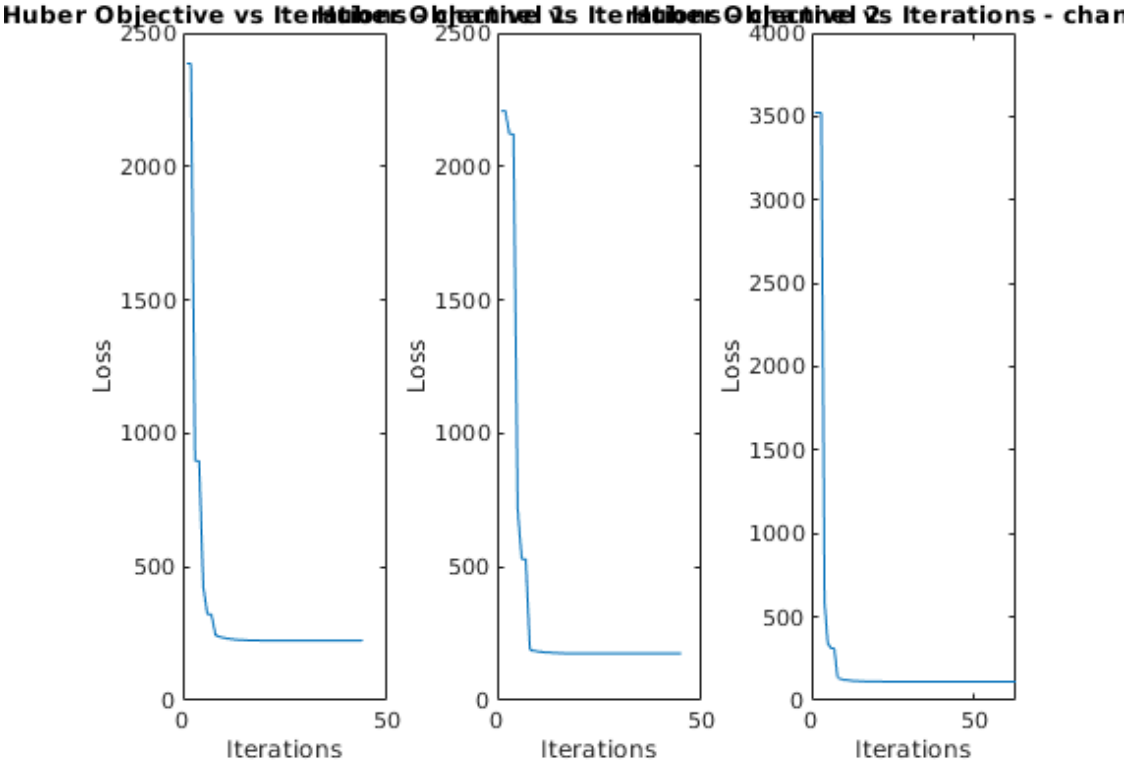


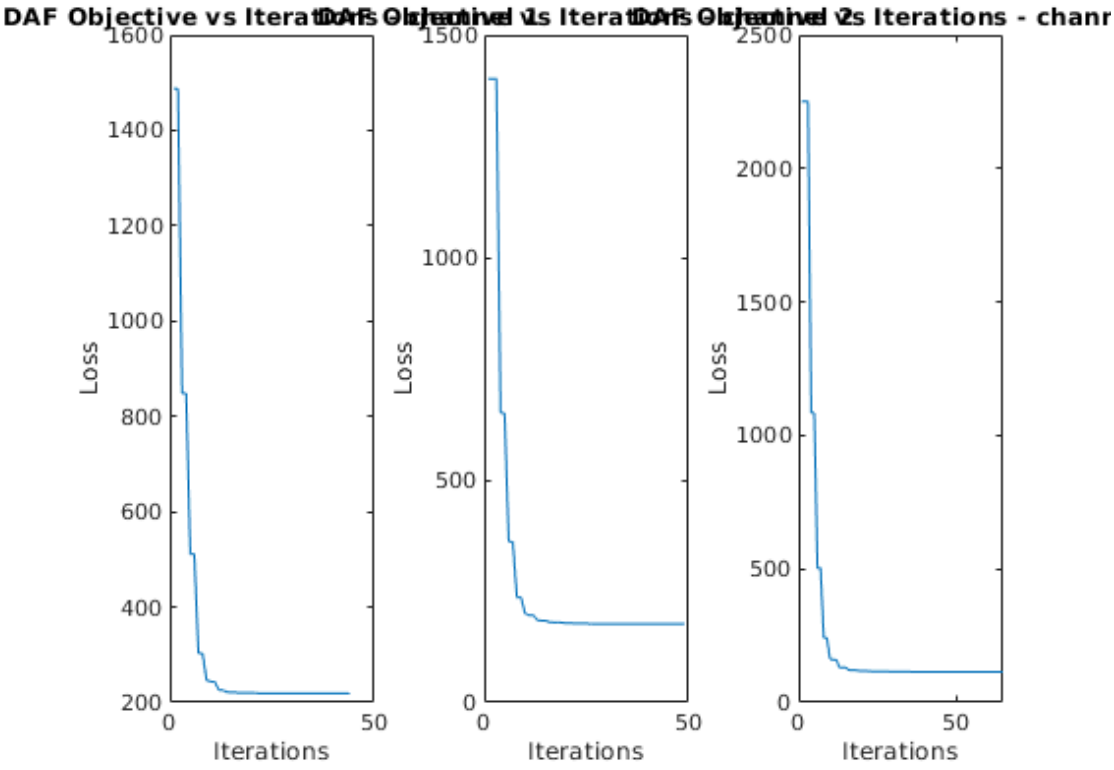
Loss vs Iterations

```
figure;

subplot(1, 3, 1);
plot(loss11);
xlabel("Iterations");
ylabel("Loss");
title("Quadratic Objective vs Iterations - channel 1");

subplot(1, 3, 2);
plot(loss12);
xlabel("Iterations");
ylabel("Loss");
title("Quadratic Objective vs Iterations - channel 2");

subplot(1, 3, 3);
plot(loss13);
xlabel("Iterations");
ylabel("Loss");
title("Quadratic Objective vs Iterations - channel 3");

figure;

subplot(1, 3, 1);
plot(loss21);
```

```
xlabel("Iterations");
ylabel("Loss");
title("Huber Objective vs Iterations - channel 1");

subplot(1, 3, 2);
plot(loss22);
xlabel("Iterations");
ylabel("Loss");
title("Huber Objective vs Iterations - channel 2");

subplot(1, 3, 3);
plot(loss23);
xlabel("Iterations");
ylabel("Loss");
title("Huber Objective vs Iterations - channel 3");

figure;

subplot(1, 3, 1);
plot(loss31);
xlabel("Iterations");
ylabel("Loss");
title("DAF Objective vs Iterations - channel 1");

subplot(1, 3, 2);
plot(loss32);
xlabel("Iterations");
ylabel("Loss");
title("DAF Objective vs Iterations - channel 2");

subplot(1, 3, 3);
plot(loss33);
xlabel("Iterations");
ylabel("Loss");
title("DAF Objective vs Iterations - channel 3");
```


Loss
Iterations
0
1000
2000
3000
4000
5000
6000
50
100
500
1500
2500
3500
4500


Loss
Iterations
0
500
1000
1500
2000
2500
3000
3500
4000
50




```
toc;
```

```
Elapsed time is 14.961570 seconds.
```

Published with MATLAB® R2019b