Национальный Исследовательский Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №6

по дисциплине «Цифровая обработка сигналов»

Выполнил:Ларочкин Г.И

Группа: P3400

Преподаватель: Тропченко А.А.

Санкт-Петербург

2020 г.

## Постановка задачи

1. Используя данные таблицы 1, создайте функции, реализующие каждый из представленных видов шумов. Программный код функций с комментариями поместите в отчет.

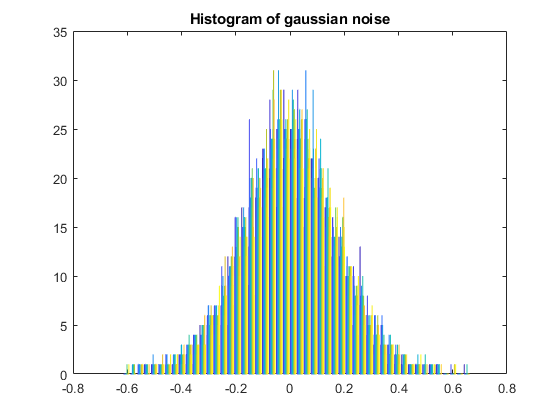
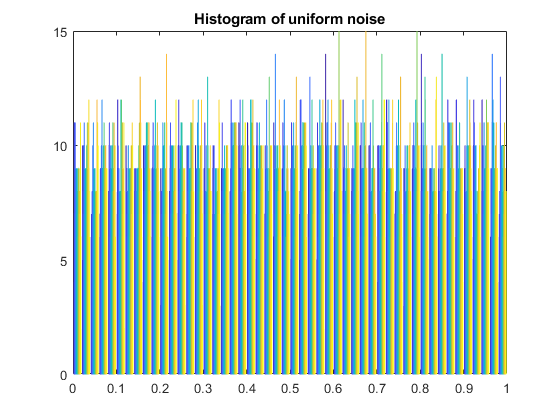
|  |  |  |  |
| --- | --- | --- | --- |
| Тип шума | Генератор шума (z) | Параметры для данной работы | |
| a | b |
| Равномерный | функция rand | - | - |
| Гауссов | функция a + b\*randn | 0 | 0.15 |
| Логарифмически нормальный |  | 1 | 0.25 |
| Реллея |  | 0 | 1 |
| Экспоненциальный |  | 1 | - |
| Эрланга | z = E1 + E2 + … + Eb,  где En = | 2 | 5 |

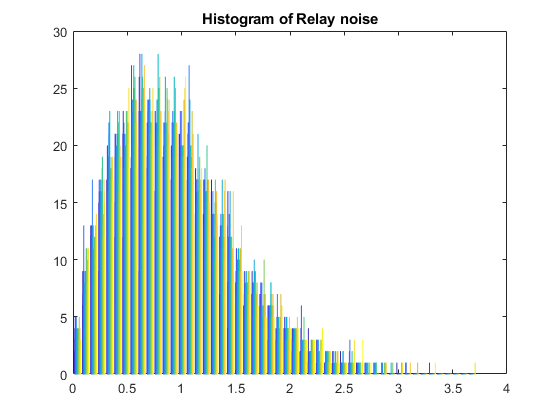
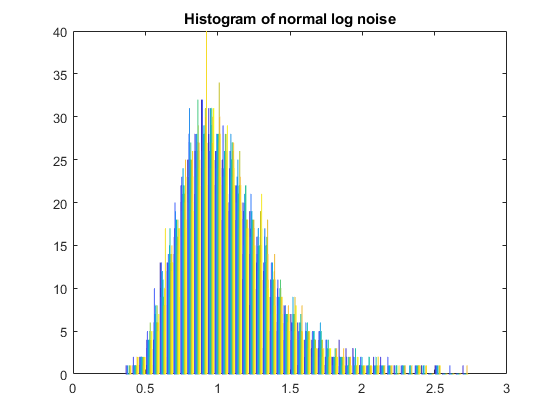
1. Осуществите генерацию каждого типа шума размером 256x256 при помощи созданных функций. Для всех шумов, кроме шума Соль и перец создайте гистограмму и поместите их в отчет. Количество корзин гистограммы должно иметь значение, равное 50.
2. Используя созданные функции и полученное для данной лабораторной работы изображение осуществите зашумление полученного изображения при помощи всех видов шумов, представленных в таблице 1. Каждый полученный файл сохраните в виде отдельного изображения. Полученные изображения для каждого вида шума поместите в отчет, указав при этом вид шума, которому изображение соответствует.
3. Создайте функции, реализующие фильтрацию изображения каждым фильтром, формула которого представлена в таблице 2. При реализации конкретного фильтра используйте только его формулу. Код программы с комментариями поместите в отчет.
4. Каждое изображение, полученное в задании 3, подвергните фильтрации всеми фильтрами, используя созданные в задании № 4 функции. Используя функцию корреляции Пирсона corr2, определите качество каждого восстановленного изображения, сравнив его с оригиналом, выданным для выполнения данной работы. Код программы с комментариями поместите в отчет.

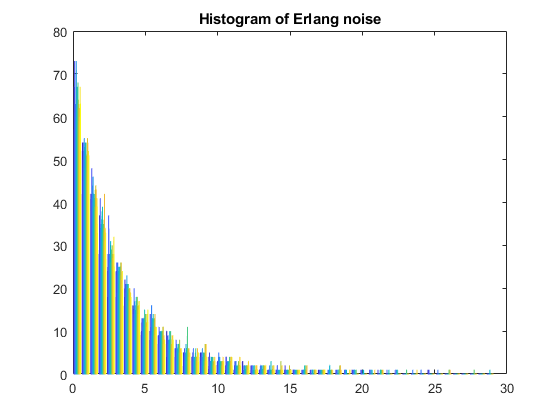
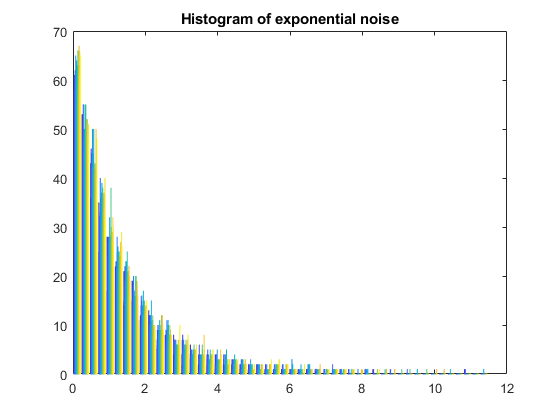
# Решение задачи

## Реализация функций шума

Было реализовано 6 видов шумов: равномерный, Гауссов, логарифмически нормальный, Реллэя, экспоненциальный и Эрланга. Ниже приведены гистограммы распределений каждого из шумов на размере матрицы 256x256:







Part1.m:

noise = uniform(256, 256);

figure, hist(noise, 50);

title('Histogram of uniform noise');

noise = gaussian(256, 256);

figure, hist(noise, 50);

title('Histogram of gaussian noise');

noise = logn(256, 256);

figure, hist(noise, 50);

title('Histogram of normal log noise');

noise = relay(256, 256);

figure, hist(noise, 50);

title('Histogram of Relay noise');

noise = exponential(256, 256);

figure, hist(noise, 50);

title('Histogram of exponential noise');

noise = erlang(256, 256);

figure, hist(noise, 50);

title('Histogram of Erlang noise');

function x = uniform(n, m)

x = rand(n, m);

end

function x = gaussian(n, m)

x = 0.15 \* randn(n, m);

end

function x = logn(n, m)

x = exp(0.25 \* randn(n, m));

end

function x = relay(n, m)

x = sqrt(-log(1 - rand(n, m)));

end

function x = exponential(n, m)

x = -log(1 - rand(n, m));

end

function x = erlang(n, m)

x = -0.5 \* 5 \* log(1 - rand(n, m));

end

## Зашумление исходного изображения

На исходное изображение был наложен каждый из шумов.



Рис. 4 – исходное изображение

Рис. 5 – зашумление равномерным и Гауссовым шумом

Рис. 6 – зашумление логарифмически нормальным и Реллеявским шумом

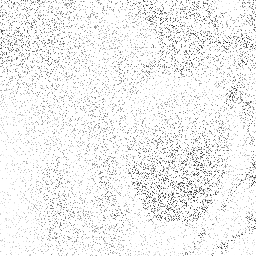
 

Рис. 7 – зашумление экспоненциальным и шумом Эрланга

Part2.m:

size = 256;

i = imread('resources/gray.jpg');

i = imresize(i, 0.5); % 512x512 to 256x256

i = im2double(i);

n = i + uniform(size, size);

imwrite(n, 'resources/uniform/img\_uniform\_noise.png');

n = i + gaussian(size, size);

imwrite(n, 'resources/gaussian/img\_gaussian\_noise.png');

n = i + logn(size, size);

imwrite(n, 'resources/log/img\_log\_noise.png');

n = i + relay(size, size);

imwrite(n, 'resources/relay/img\_relay\_noise.png');

n = i + exponential(size, size);

imwrite(n, 'resources/exp/img\_exp\_noise.png');

n = i + erlang(size, size);

imwrite(n, 'resources/erlang/img\_erlang\_noise.png');

## Реализация фильтрующих функций

Было реализовано девять фильтрующих функций, формулы представлены на рисунке 8

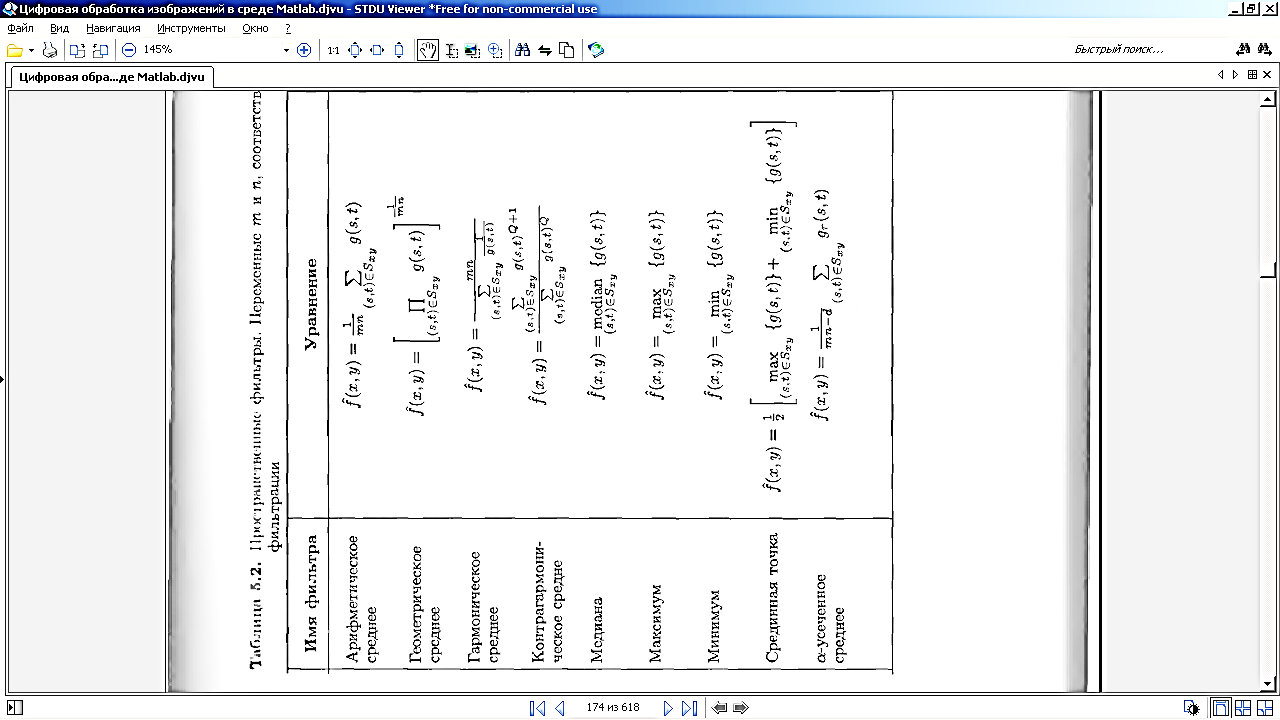


Рис. 8 – фильтрующие функции

Part3.m:

function g = imfilter3x3(f, func)

f = im2double(f);

g = zeros(size(f));

for i = 2:1:size(f,1)-1

for j = 2:1:size(f,2)-1

mask = f(i-1:i+1,j-1:j+1);

g(i,j) = func(mask);

end

end

end

function g = mean\_filter(f)

func = @(x) sum(sum(x)) / 9;

g = imfilter3x3(f, func);

end

function g = geometric\_mean\_filter(f)

func = @(x) sum(prod(x(:))) ^ (1 / 9);

g = imfilter3x3(f, func);

end

function g = harmonic\_mean\_filter(f)

func = @(x) 9 / sum(sum(ones(3, 3) ./ x));

g = imfilter3x3(f, func);

end

function g = contrharmonic\_mean\_filter(f, q)

func = @(x) sum(sum(x.^(q+1))) / sum(sum(x.^q));

g = imfilter3x3(f, func);

end

function g = median\_filter(f)

func = @(x) median(x(:));

g = imfilter3x3(f, func);

end

function g = max\_filter(f)

func = @(x) max(x(:));

g = imfilter3x3(f, func);

end

function g = min\_filter(f)

func = @(x) min(x(:));

g = imfilter3x3(f, func);

end

function g = avg\_filter(f)

func = @(x) (max(x(:)) + min(x(:))) / 2;

g = imfilter3x3(f, func);

end

function g = alpha\_truncated\_filter(f, d)

f = im2double(f);

g = zeros(size(f));

for i = 2:1:size(f,1)-1

for j = 2:1:size(f,2)-1

mask = f(i-1:i+1,j-1:j+1);

x = sort(mask(:));

x = x(1+d/2:end);

x = x(1:end - d/2);

g(i,j) = sum(x) / (9 - d);

end

end

end

## Фильтрация зашумлённых изображений

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Равномерный | Гауссов | Логарифмически нормальный | Реллея | Экспоненциальный | Эрланга |
| Арифметическое среднее | 0.1233 | 0.1985 | 0.0027 | 0.0677 | 0.1002 | 0.1674 |
| Геометрическое среднее | 0.1381 | 0.2013 | 0.0035 | 0.0801 | 1.1205 | 0.2074 |
| Гармоническое среднее | 0.1512 | 0.1994 | 0.0043 | 0.0938 | 1.1392 | 0.2458 |
| Контргармоническое среднее | 0.0630 | 0.1810 | 0.0000 | 0.0303 | 0.0298 | 0.0530 |
| Медиана | 0.1125 | 0.1962 | 0.0073 | 0.0246 | 0.0678 | 0.0308 |
| Максимумы | 0.0036 | 0.1738 | 0.0078 | 0.0078 | 0.0077 | 0.0078 |
| Минимумы | 0.1759 | 0.1849 | 0.0525 | 0.1490 | 0.1690 | 0.3133 |
| Срединная точка | 0.1264 | 0.1915 | 0.0240 | 0.1062 | 0.1217 | 0.2413 |
| α-усеченное среднее | 0.1177 | 0.1977 | 0.0060 | 0.0400 | 0.0826 | 0.0882 |

Таблица 1 – корреляция Пирсона для каждого фильтра с каждым зашумленным изображением

В таблице 1 приведены значения корреляции Пирсона отфильтрованных изображений. Исходя из этой таблицы, можно сделать вывод, что для данного изображения хорошо работают фильтры, которые фильтруют тёмные цвета, например, фильтр минимума. Это верно, ведь исходное изображение содержит большое количество тёмных цветов.

Плохие результаты выдают фильтры максимумы и контргармоническое среднее (использовалось Q=2). Эти фильтры хорошо фильтруют светлые изображения. Ниже представлены лучшие отфильтрованные изображения.

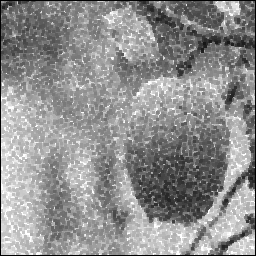
 

Рис. 9 – равномерный шум, фильтр минимума и шум Гаусса, геометрически средний фильтр

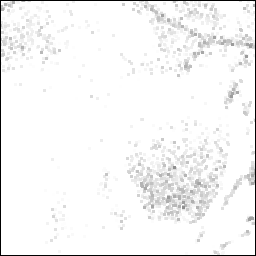
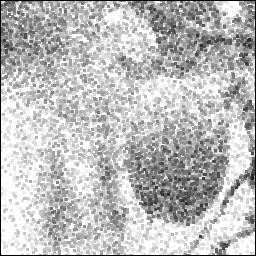
 

Рис. 10 – логарифмически нормальный шум, фильтр минимума и шум Реллея, фильтр минимума

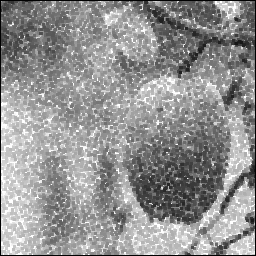
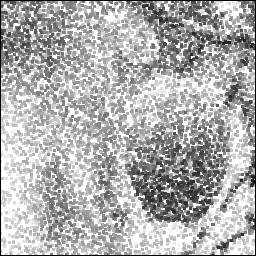
 

Рис. 11 –экспоненциальный шум, фильтр минимума и шум Эрланга, фильтр минимума

Part3.m:

size = 256;

sources = ["uniform", "gaussian", "log", "relay", "exp", "erlang"];

is = imread('resources/gray.jpg');

is = imresize(i, 0.5); % 512x512 to 256x256

is = im2double(i);

for i = 1:length(sources)

name = sources(i);

dir = sprintf('resources/%s', name);

str = sprintf('%s/img\_%s\_noise.png', dir, name);

i = imread(str);

i = im2double(i);

ir = mean\_filter(i);

str = sprintf('%s/img\_mean\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise mean filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = geometric\_mean\_filter(i);

str = sprintf('%s/img\_geometric\_mean\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise geometric mean filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = harmonic\_mean\_filter(i);

str = sprintf('%s/img\_harmonic\_mean\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise harmonic mean filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = contrharmonic\_mean\_filter(i, 2);

str = sprintf('%s/img\_contrharmonic\_mean\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise contrharmonic mean filter with Q=2 Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = median\_filter(i);

str = sprintf('%s/img\_median\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise median filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = max\_filter(i);

str = sprintf('%s/img\_max\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise max filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = min\_filter(i);

str = sprintf('%s/img\_min\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise min filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = avg\_filter(i);

str = sprintf('%s/img\_avg\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise avg filter Pirson correlation: %.4f\n', name, corr2(is, ir));

ir = alpha\_truncated\_filter(i, 4);

str = sprintf('%s/img\_alpha\_truncated\_filter.png', dir);

imwrite(ir, str);

fprintf('%s noise alpha truncated filter with d=4 Pirson correlation: %.4f\n', name, corr2(is, ir));

end