**Open Software Project**

**경영학과 1517034 정성원**

**Assignment 2**

**1. PDF / CDF Generation**

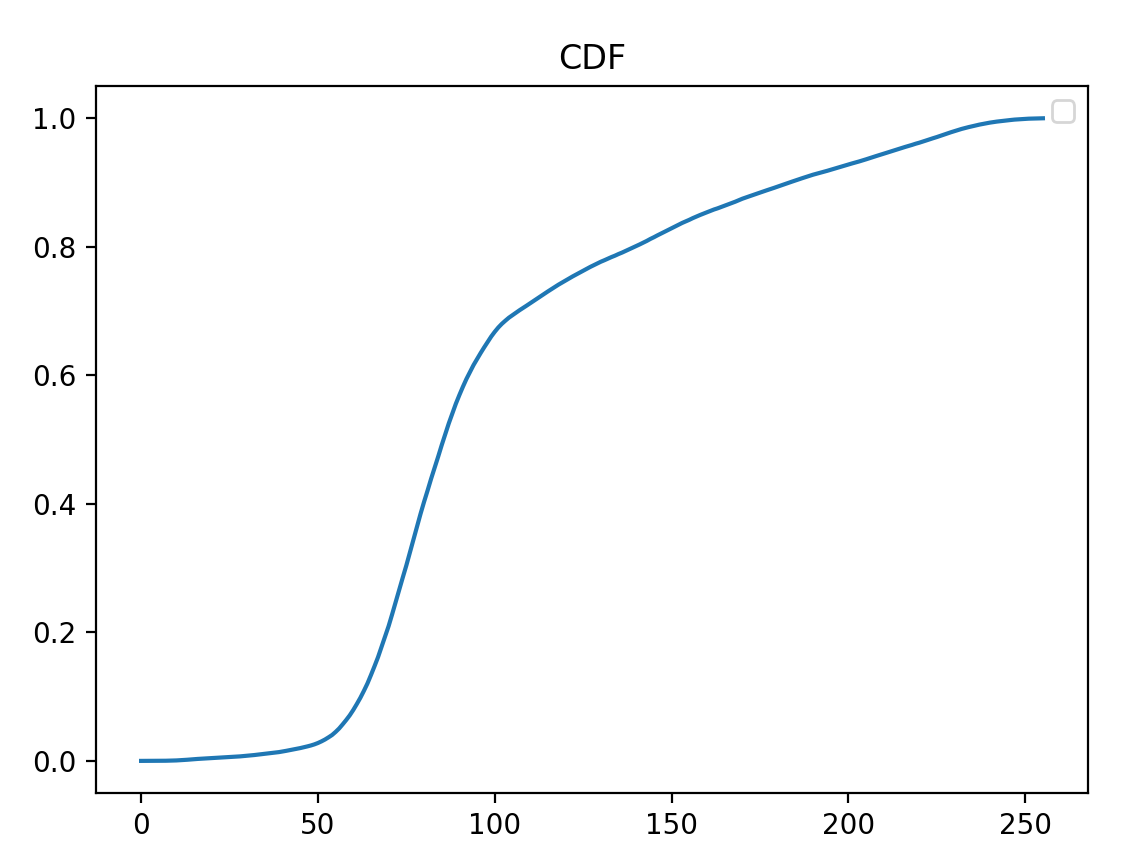
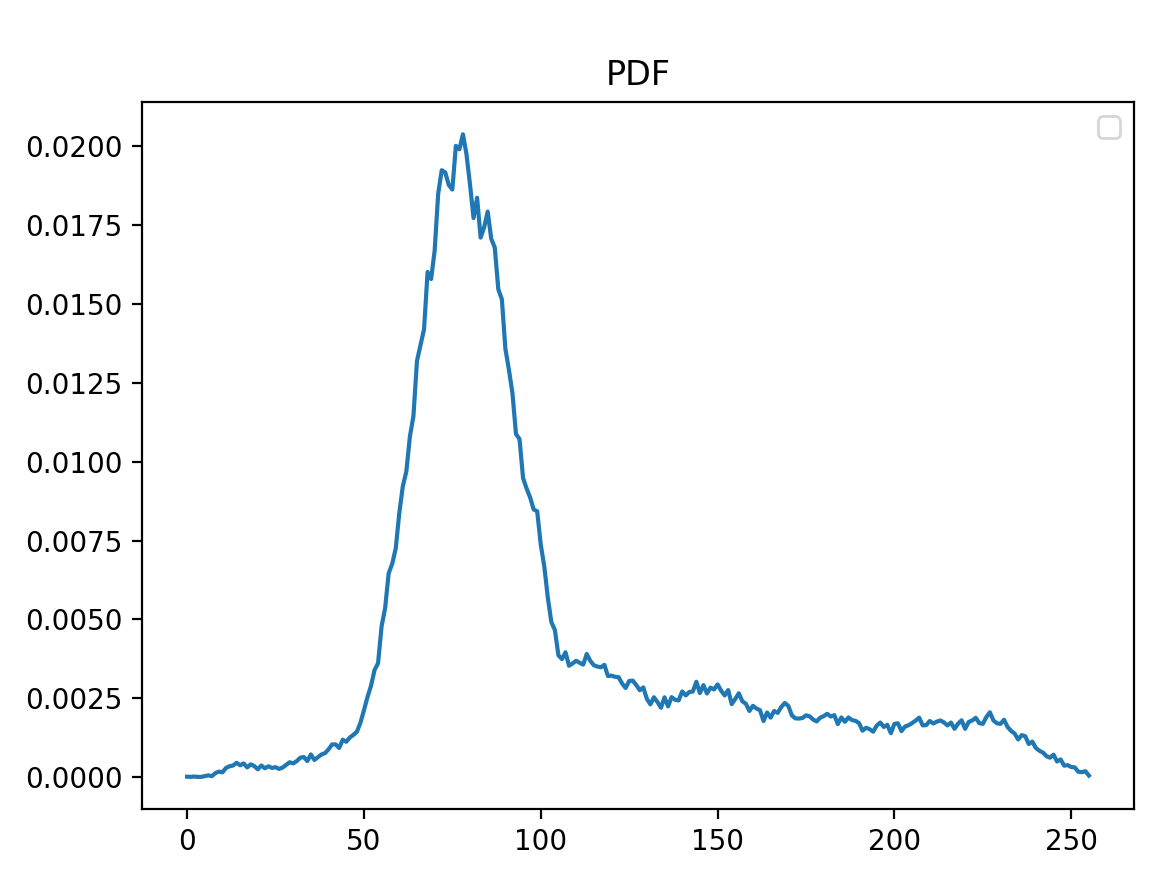
**A. Code**

**B. Result**

-Image(grayscale)



-PDF

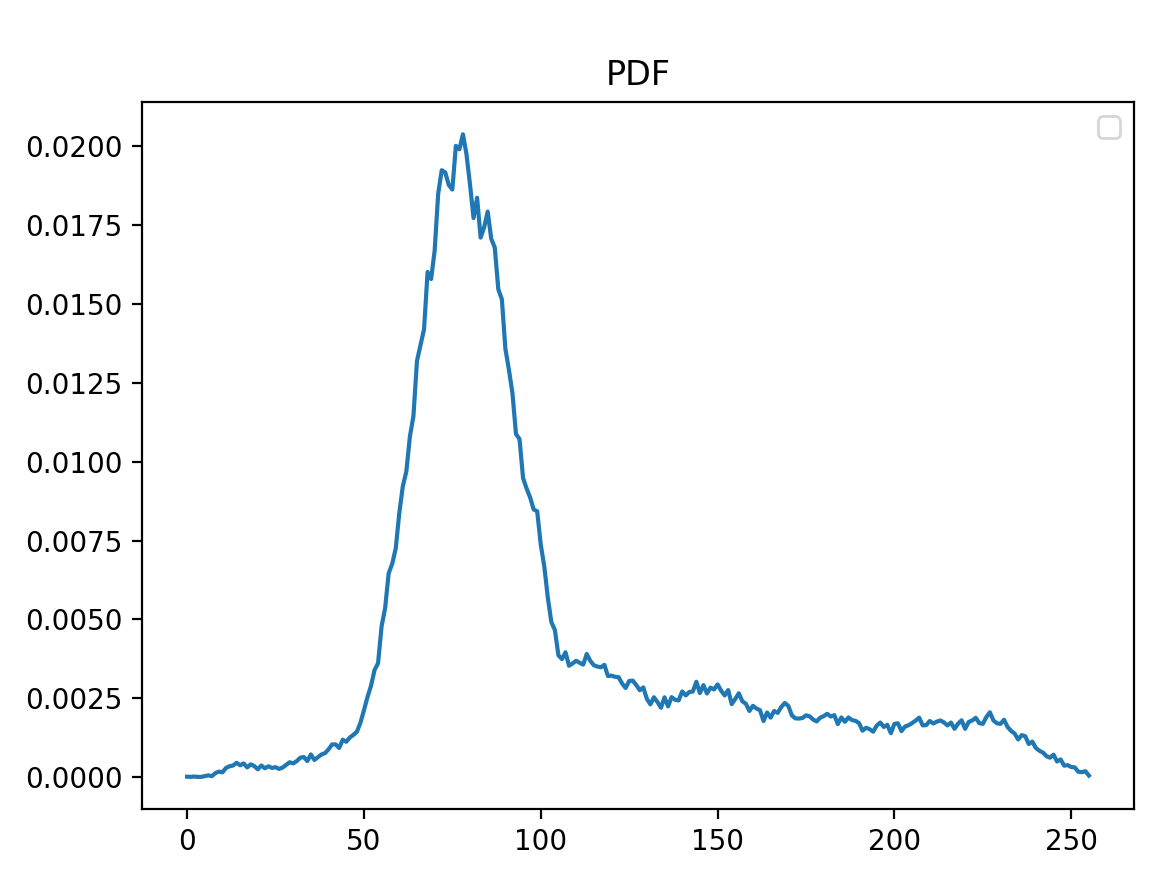


**2. Histogram Stretching**

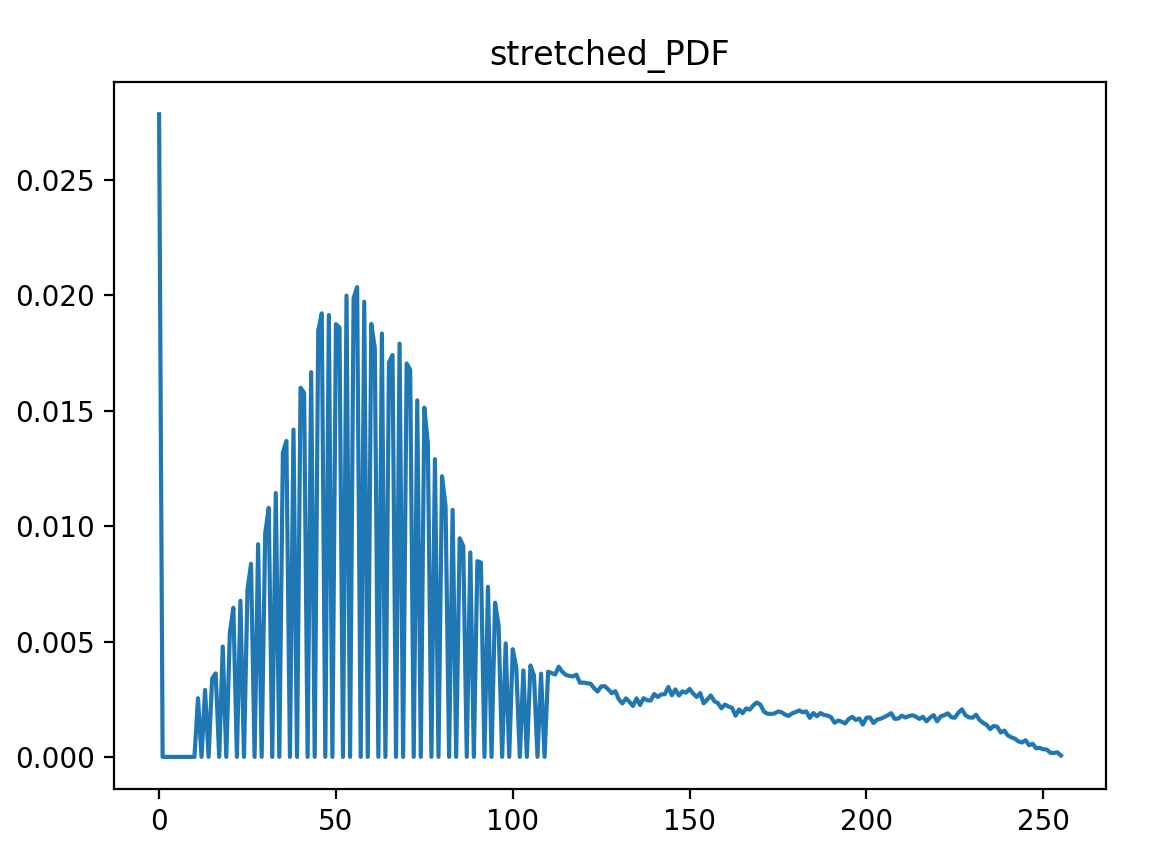
**A. Code**

**B. Result**

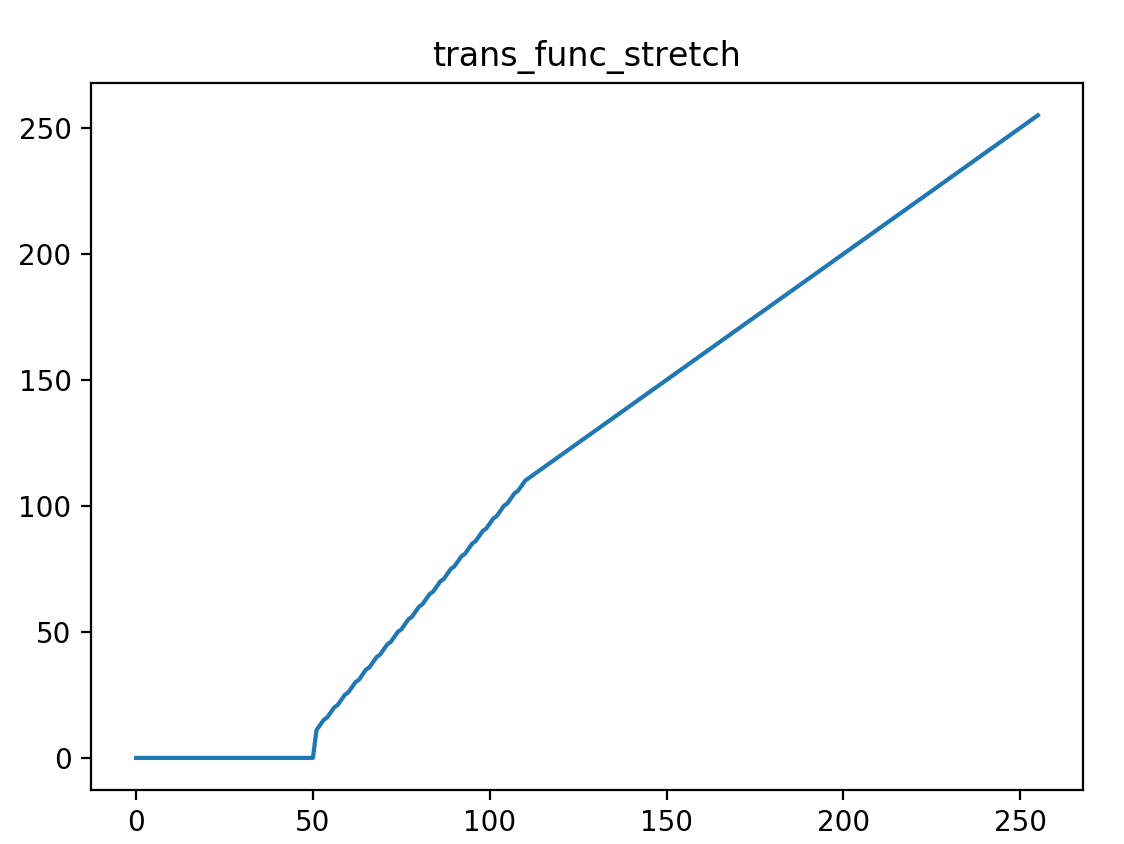
-Original Image / Histogram



-Stretched image / Histogram



-transfer function



**3. Histogram Equalization**

**1) Grayscale Image**

**A. Code**

// histogram equalization

void hist\_eq(Mat &input, Mat &equalized, G \*trans\_func, float \*CDF) {

// compute transfer function

for (int i = 0; i < L; i++)

trans\_func[i] = (G)((L - 1) \* CDF[i]);

// perform the transfer function

for (int i = 0; i < input.rows; i++)

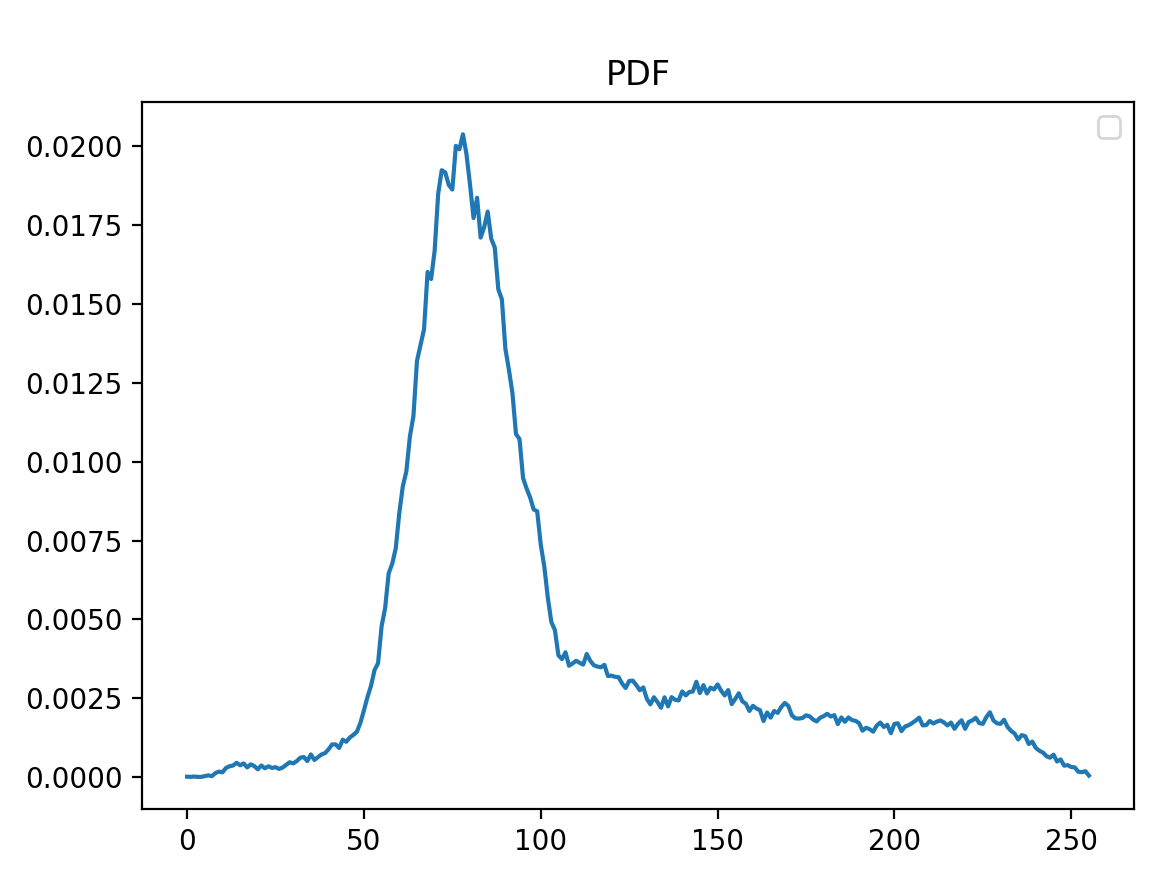
for (int j = 0; j < input.cols; j++)

equalized.at<G>(i, j) = trans\_func[input.at<G>(i, j)];

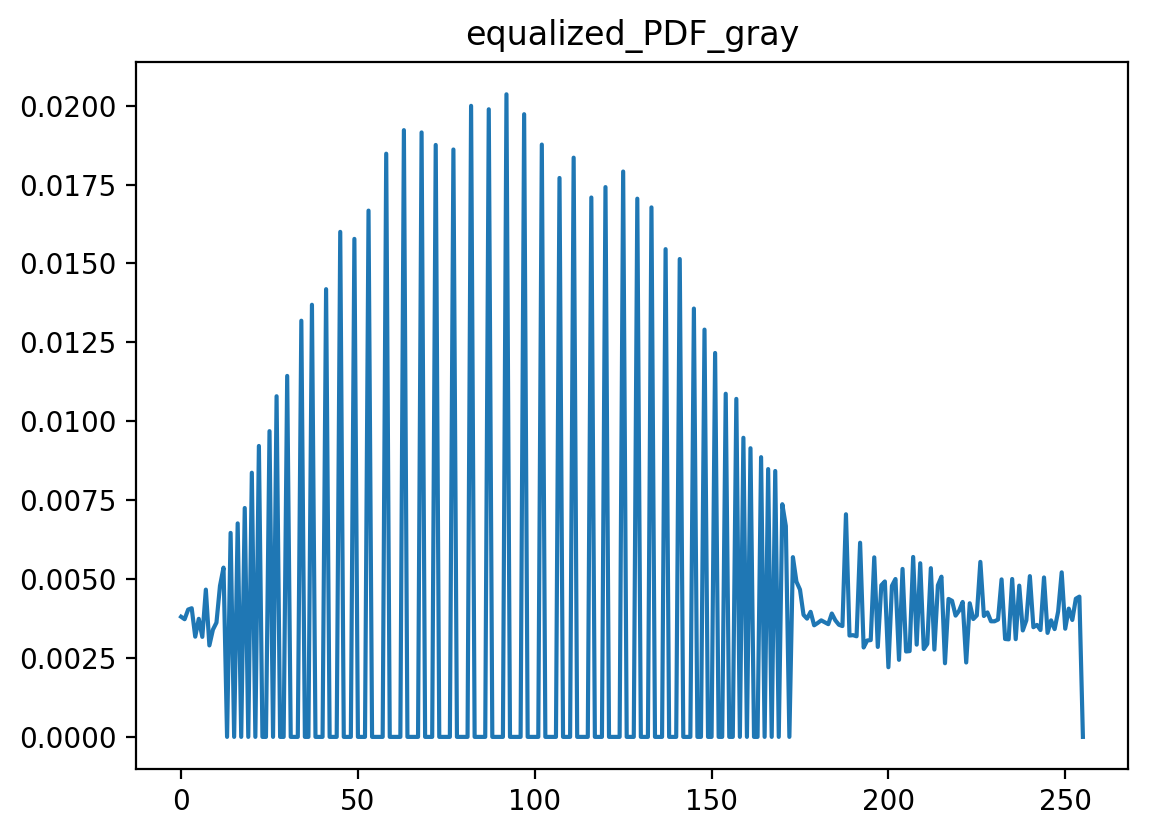
}

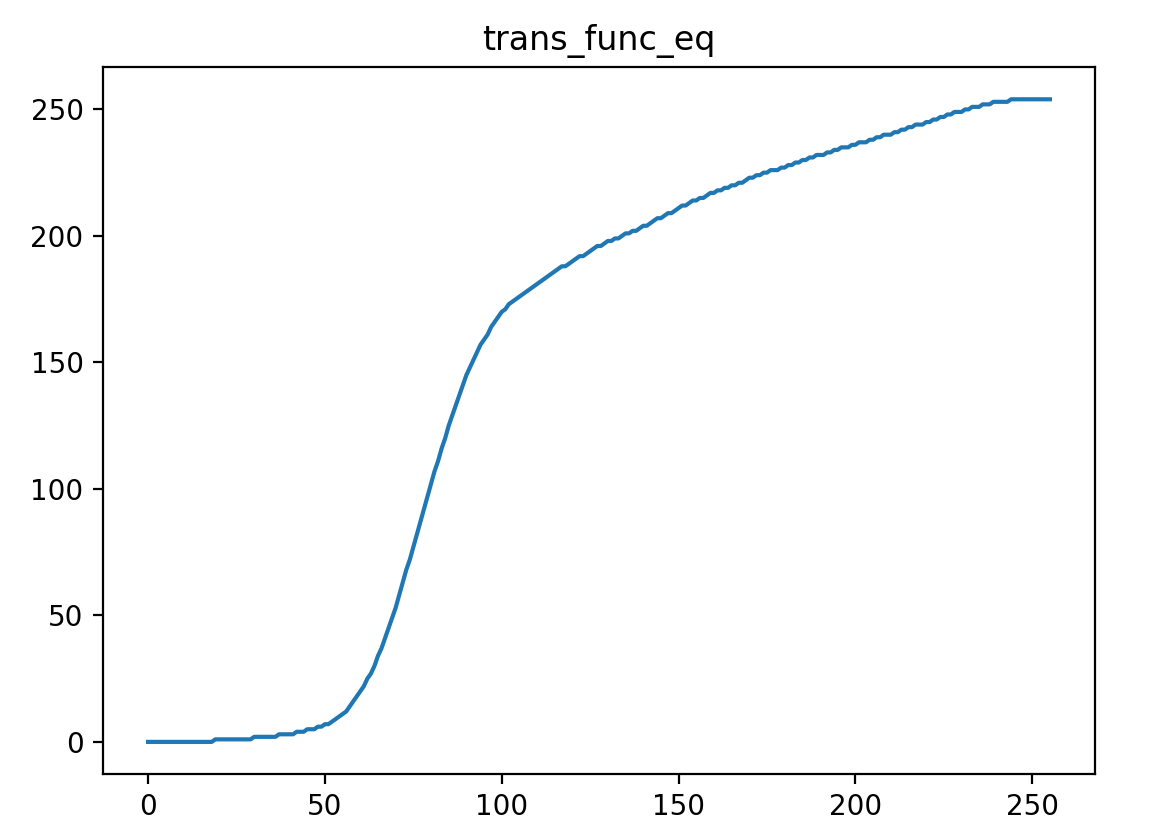
**B. Result**

-Original Image / Histogram



-Output image / Histogram





**2) RGB**

**A. Code**

// histogram equalization on 3 channel image

void hist\_eq\_Color(Mat &input, Mat &equalized, G(\*trans\_func)[3], float \*\*CDF) {

// compute transfer function

for (int i = 0; i < L; i++)

for(int k = 0; k < 3 ; k++)

trans\_func[i][k] = (G)((L - 1) \* CDF[i][k]);

// perform the transfer function

for (int i = 0; i < input.rows; i++)

for (int j = 0; j < input.cols; j++)

for (int k =0; k < 3; k++)

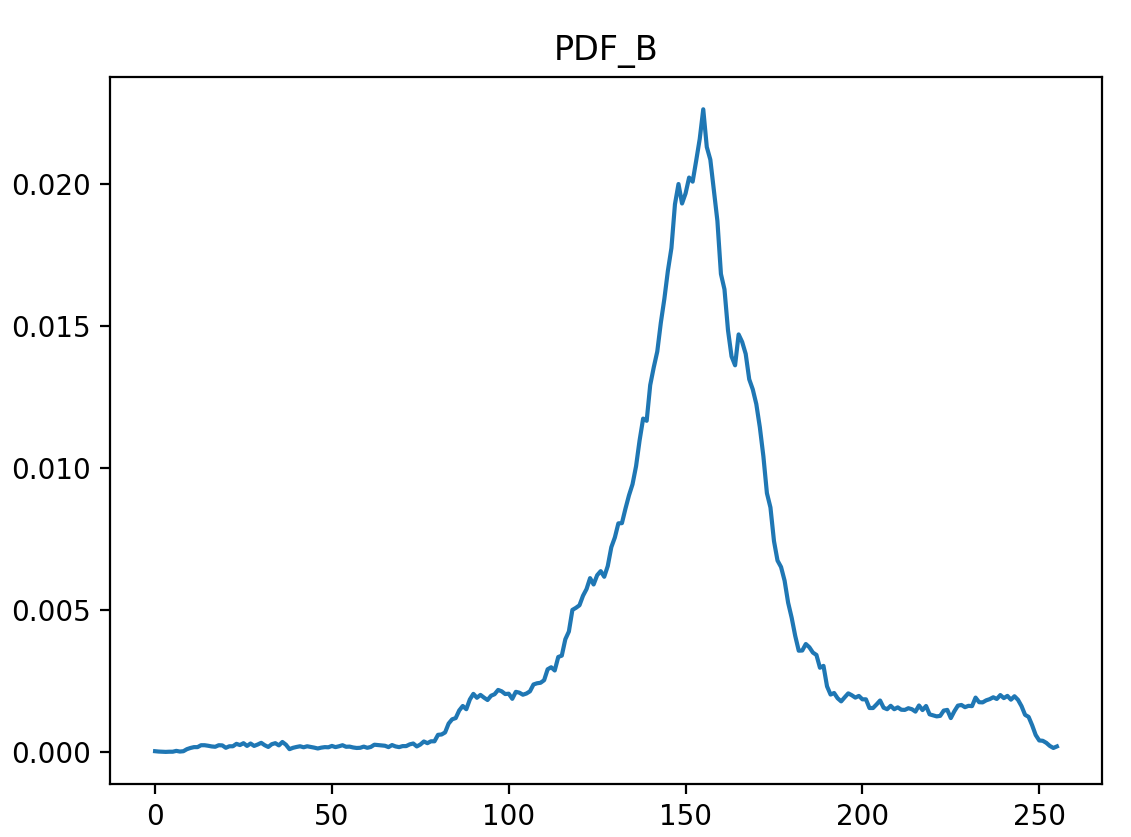
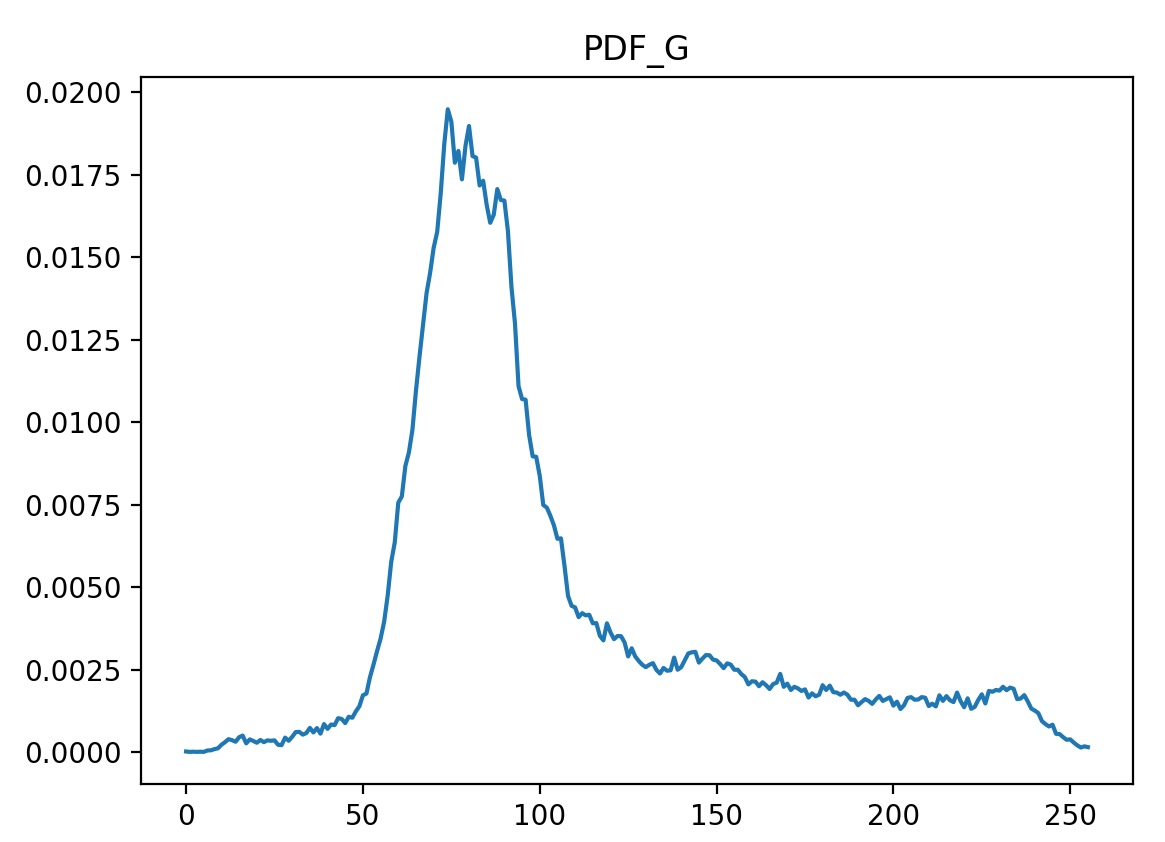
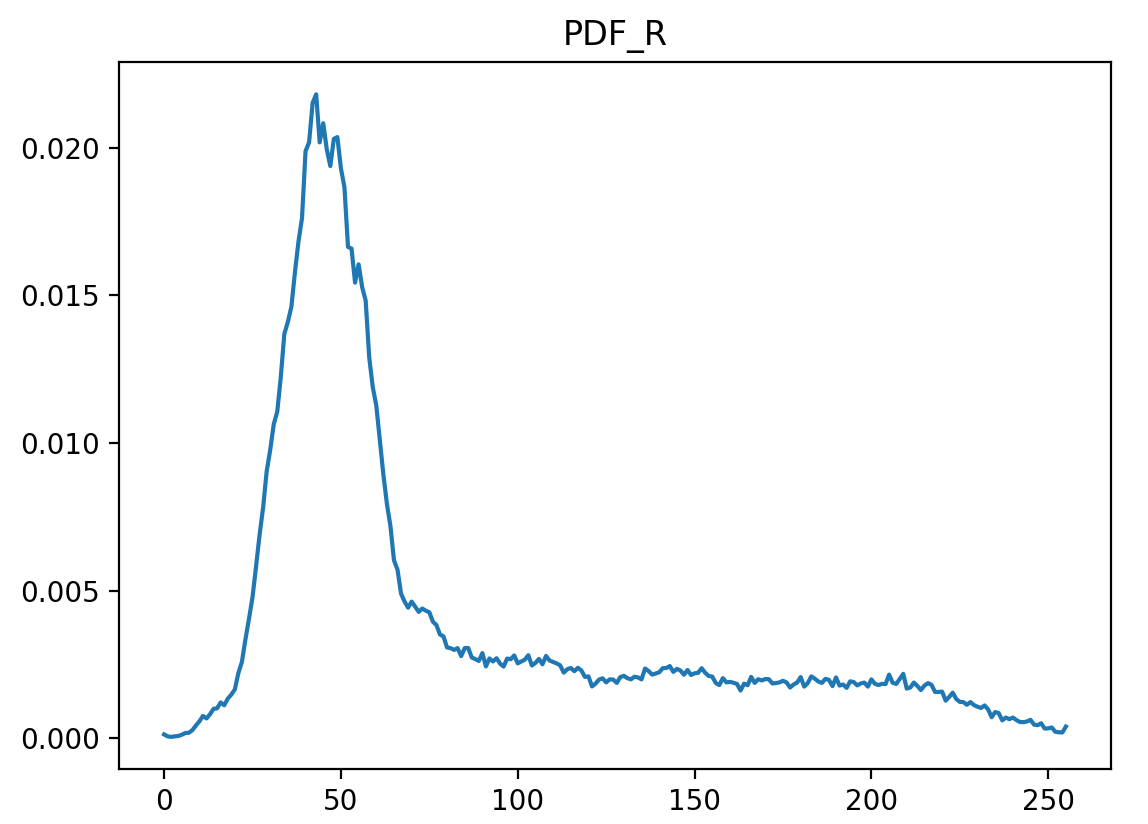
equalized.at<C>(i, j)[k] = trans\_func[input.at<C>(i, j)[k]][k];

}

**B. Result**

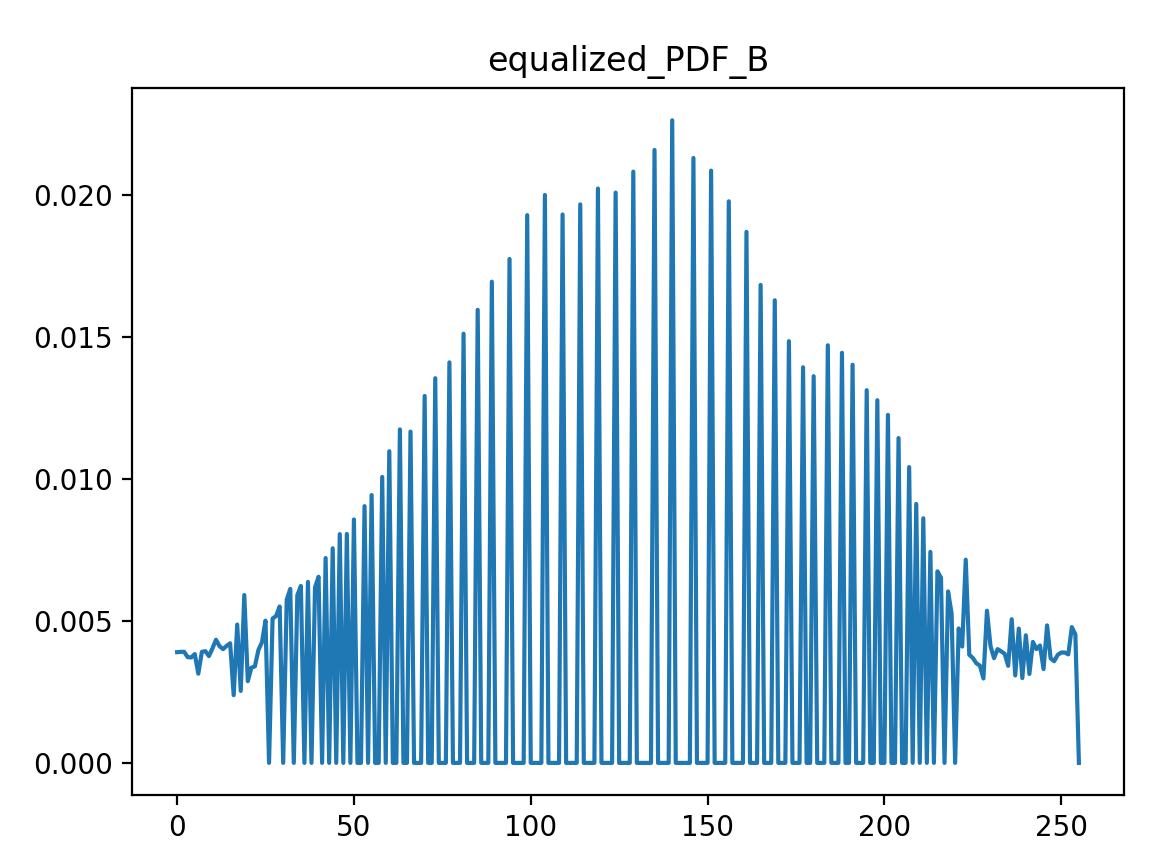
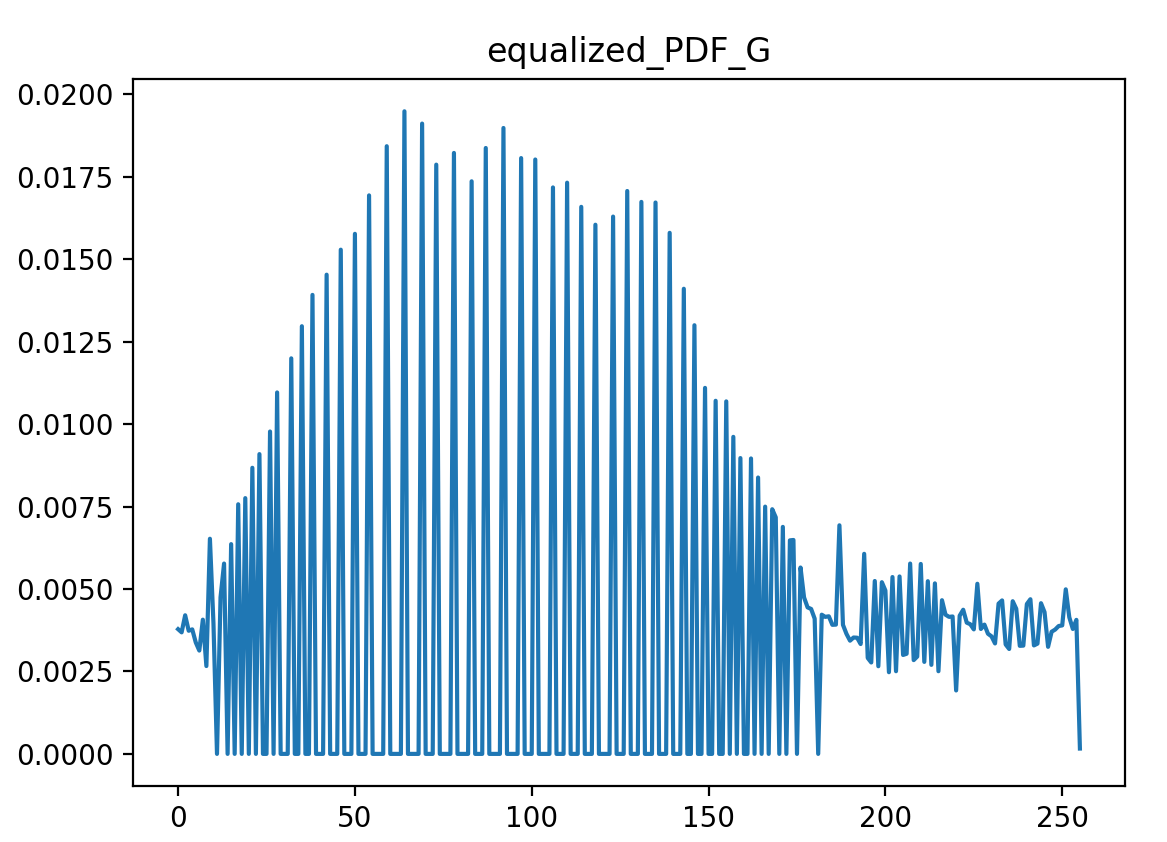
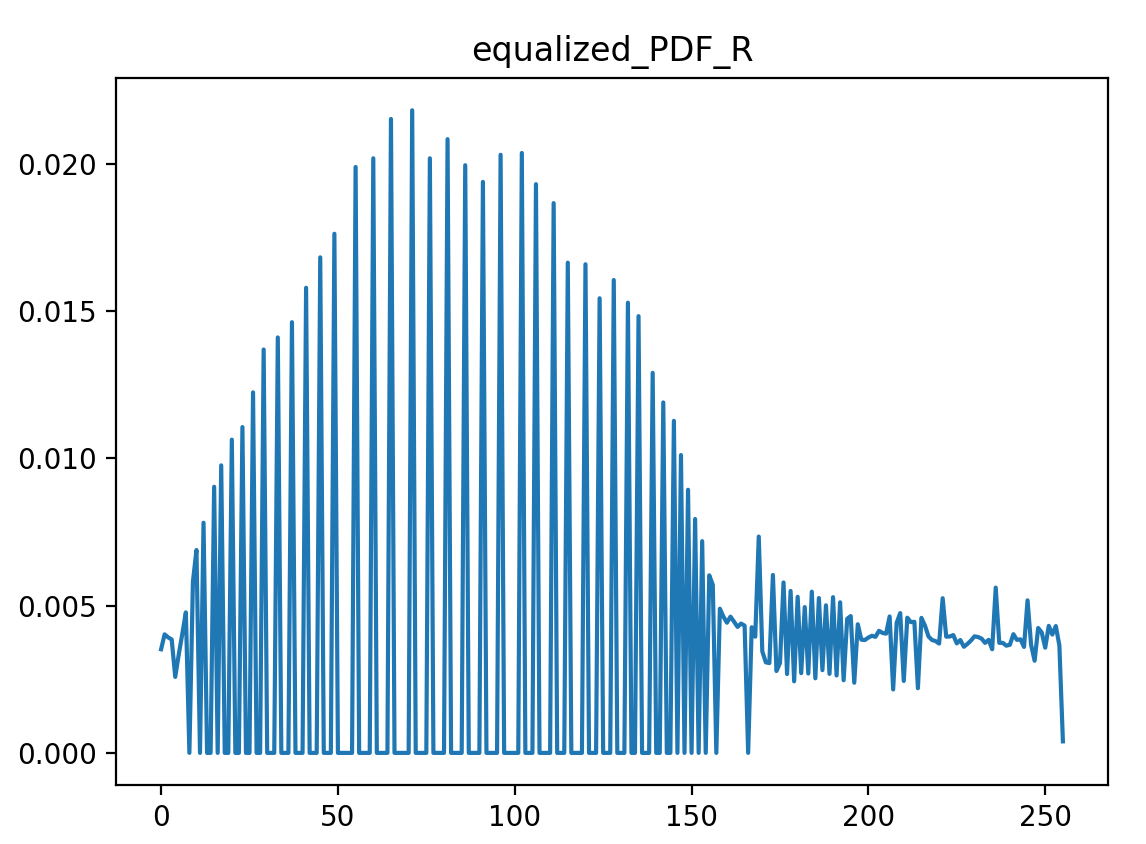
-Original image / Histogram

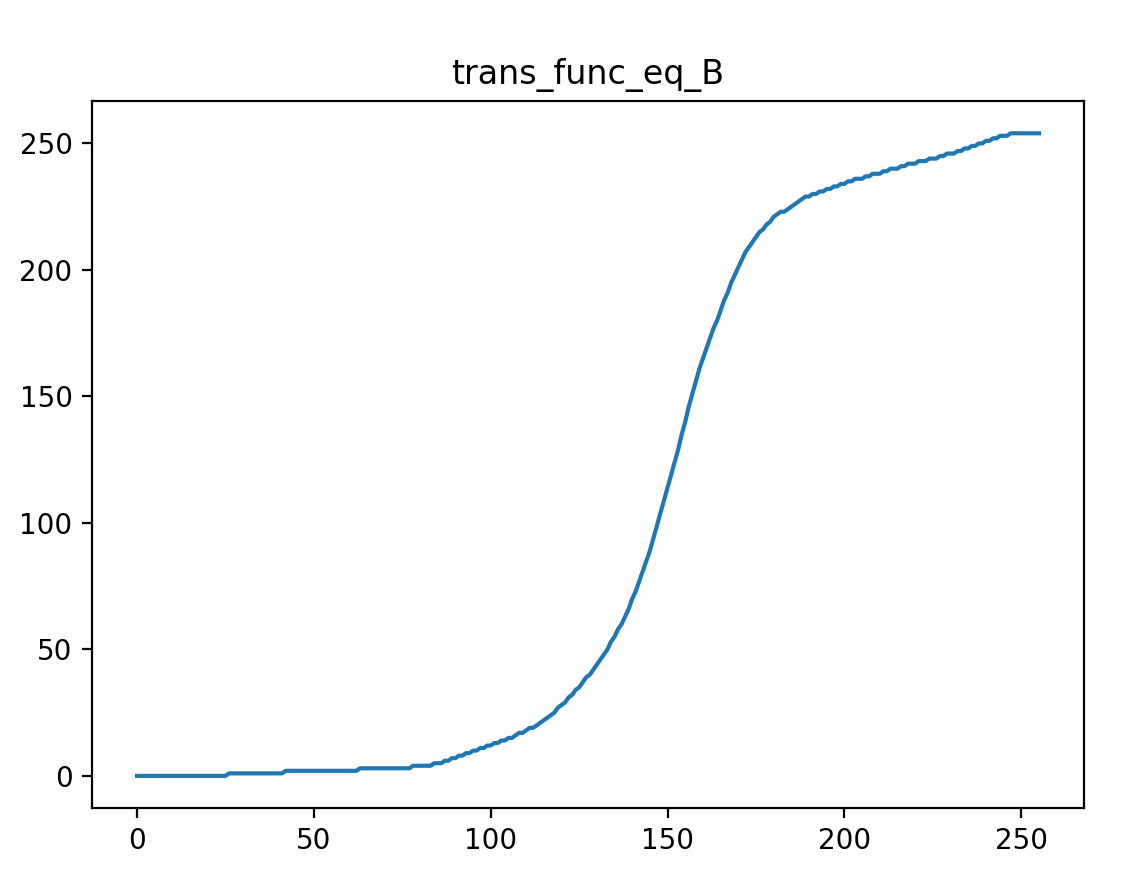
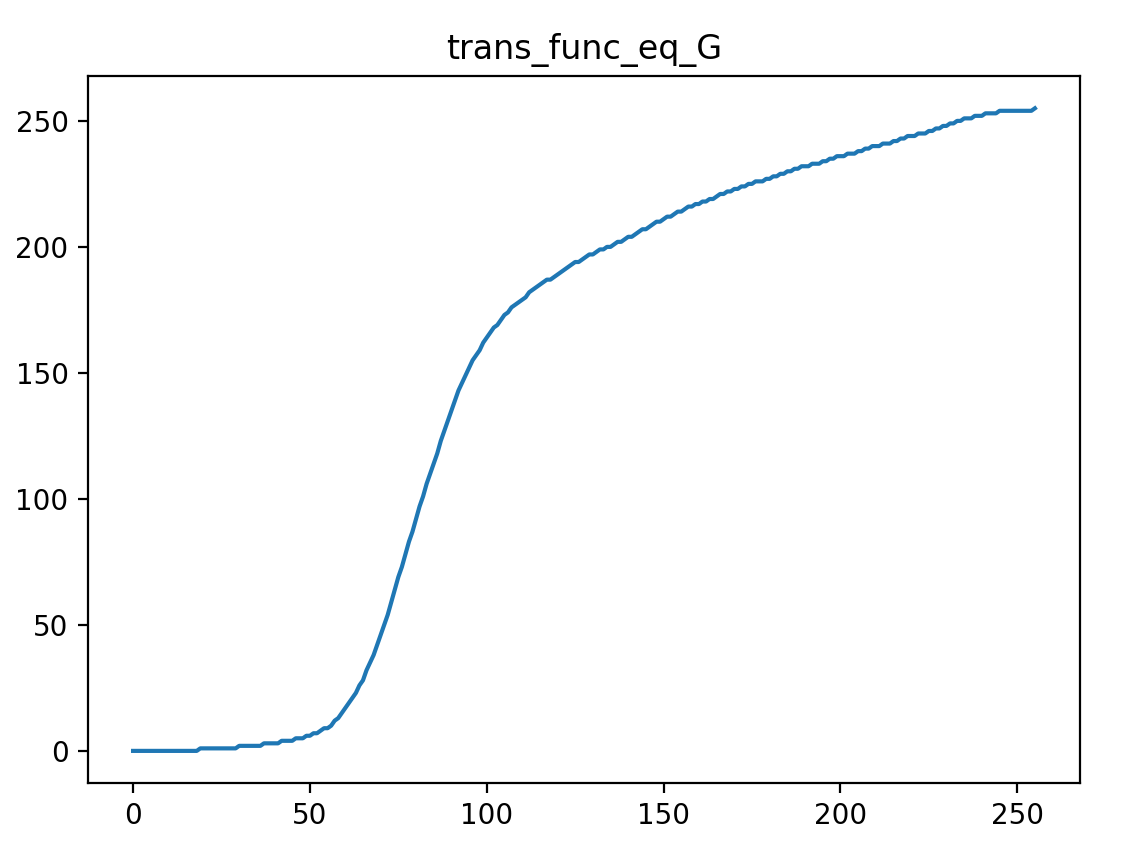
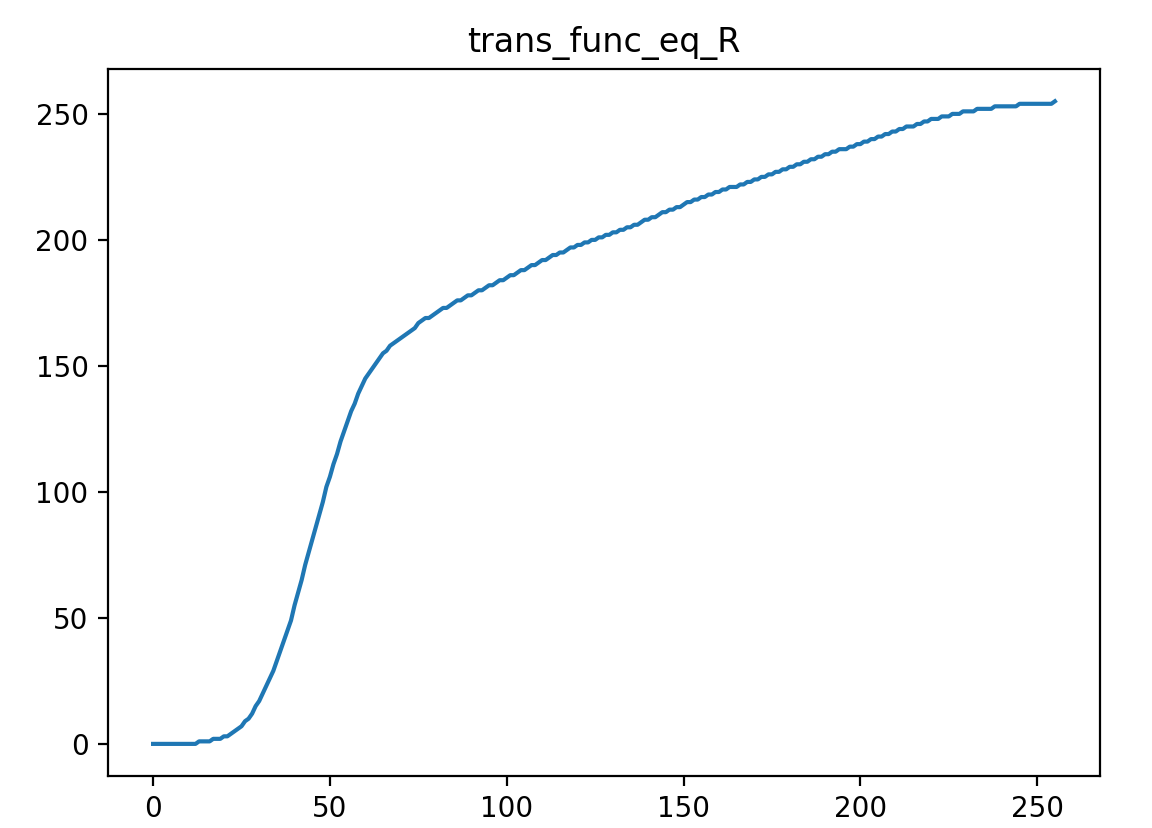




-Output image / Histogram







**3) YUV**

**A. Code**

int main() {

Mat input = imread("input.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat equalized\_YUV;

cvtColor(input, equalized\_YUV, CV\_RGB2YUV); // RGB -> YUV

// split each channel(Y, U, V)

Mat channels[3];

split(equalized\_YUV, channels);

Mat Y = channels[0]; // U = channels[1], V = channels[2]

// PDF or transfer function txt files

FILE \*f\_PDF\_R, \*f\_PDF\_G, \*f\_PDF\_B;

FILE \*f\_equalized\_PDF\_YUV\_R, \*f\_equalized\_PDF\_YUV\_G, \*f\_equalized\_PDF\_YUV\_B;

FILE \*f\_trans\_func\_eq\_YUV;

float \*\*PDF\_RGB = cal\_PDF\_RGB(input); // PDF of Input image(RGB) : [L][3]

float \*CDF\_YUV = cal\_CDF(Y); // CDF of Y channel image

fopen\_s(&f\_PDF\_R, "PDF\_R.txt", "w+"); fopen\_s(&f\_PDF\_G, "PDF\_G.txt", "w+"); fopen\_s(&f\_PDF\_B, "PDF\_B.txt", "w+");

fopen\_s(&f\_equalized\_PDF\_YUV\_R, "equalized\_PDF\_YUV\_R.txt", "w+"); fopen\_s(&f\_equalized\_PDF\_YUV\_G, "equalized\_PDF\_YUV\_G.txt", "w+"); fopen\_s(&f\_equalized\_PDF\_YUV\_B, "equalized\_PDF\_YUV\_B.txt", "w+");

fopen\_s(&f\_trans\_func\_eq\_YUV, "trans\_func\_eq\_YUV.txt", "w+");

G trans\_func\_eq\_YUV[L] = { 0 }; // transfer function

// histogram equalization on Y channel

hist\_eq(Y, Y, trans\_func\_eq\_YUV, CDF\_YUV);

// merge Y, U, V channels

merge(channels, 3, equalized\_YUV);

// YUV -> RGB (use "CV\_YUV2RGB" flag)

cvtColor(equalized\_YUV, equalized\_YUV, CV\_YUV2RGB);

// equalized PDF (YUV)

float \*\*equalized\_PDF\_YUV = cal\_PDF\_RGB(equalized\_YUV);

//write files

for (int i = 0; i < L; i++) {

// write PDF of original image

fprintf(f\_PDF\_R, "%d\t%f\n", i, PDF\_RGB[i][0]); fprintf(f\_PDF\_G, "%d\t%f\n", i, PDF\_RGB[i][1]); fprintf(f\_PDF\_B, "%d\t%f\n", i, PDF\_RGB[i][2]);

// write PDF of output image

fprintf(f\_equalized\_PDF\_YUV\_R, "%d\t%f\n", i, equalized\_PDF\_YUV[i][0]); fprintf(f\_equalized\_PDF\_YUV\_G, "%d\t%f\n", i, equalized\_PDF\_YUV[i][1]); fprintf(f\_equalized\_PDF\_YUV\_B, "%d\t%f\n", i, equalized\_PDF\_YUV[i][2]);

// write transfer functions

fprintf(f\_trans\_func\_eq\_YUV, "%d\t%d\n", i, trans\_func\_eq\_YUV[i]);

}

// memory release

free(PDF\_RGB);

free(CDF\_YUV);

fclose(f\_PDF\_R); fclose(f\_PDF\_G); fclose(f\_PDF\_B);

fclose(f\_equalized\_PDF\_YUV\_R); fclose(f\_equalized\_PDF\_YUV\_G); fclose(f\_equalized\_PDF\_YUV\_B);

fclose(f\_trans\_func\_eq\_YUV);

////////////////////// Show each image ///////////////////////

namedWindow("RGB", WINDOW\_AUTOSIZE);

imshow("RGB", input);

namedWindow("Equalized\_YUV", WINDOW\_AUTOSIZE);

imshow("Equalized\_YUV", equalized\_YUV);

//////////////////////////////////////////////////////////////

waitKey(0);

return 0;

}

// histogram equalization

void hist\_eq(Mat &input, Mat &equalized, G \*trans\_func, float \*CDF) {

// compute transfer function

for (int i = 0; i < L; i++)

trans\_func[i] = (G)((L - 1) \* CDF[i]);

// perform the transfer function

for (int i = 0; i < input.rows; i++)

for (int j = 0; j < input.cols; j++)

equalized.at<G>(i, j) = trans\_func[input.at<G>(i, j)];

}

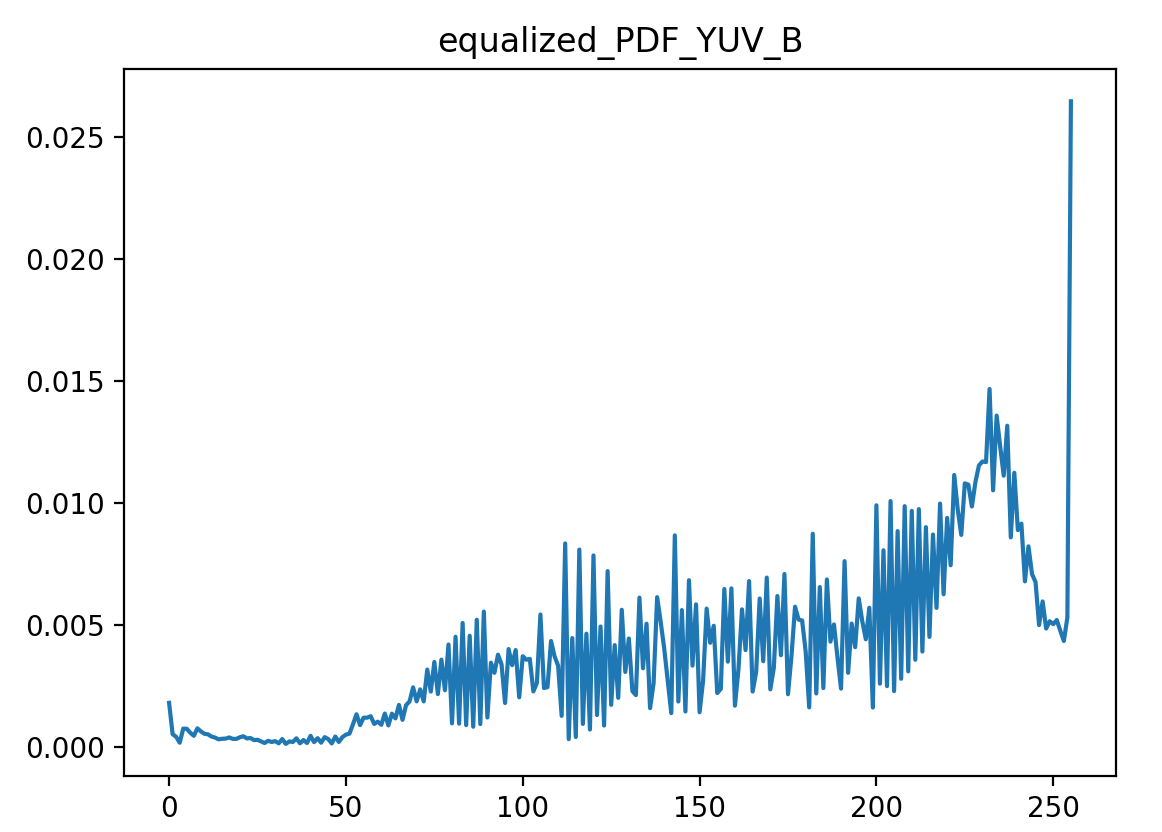
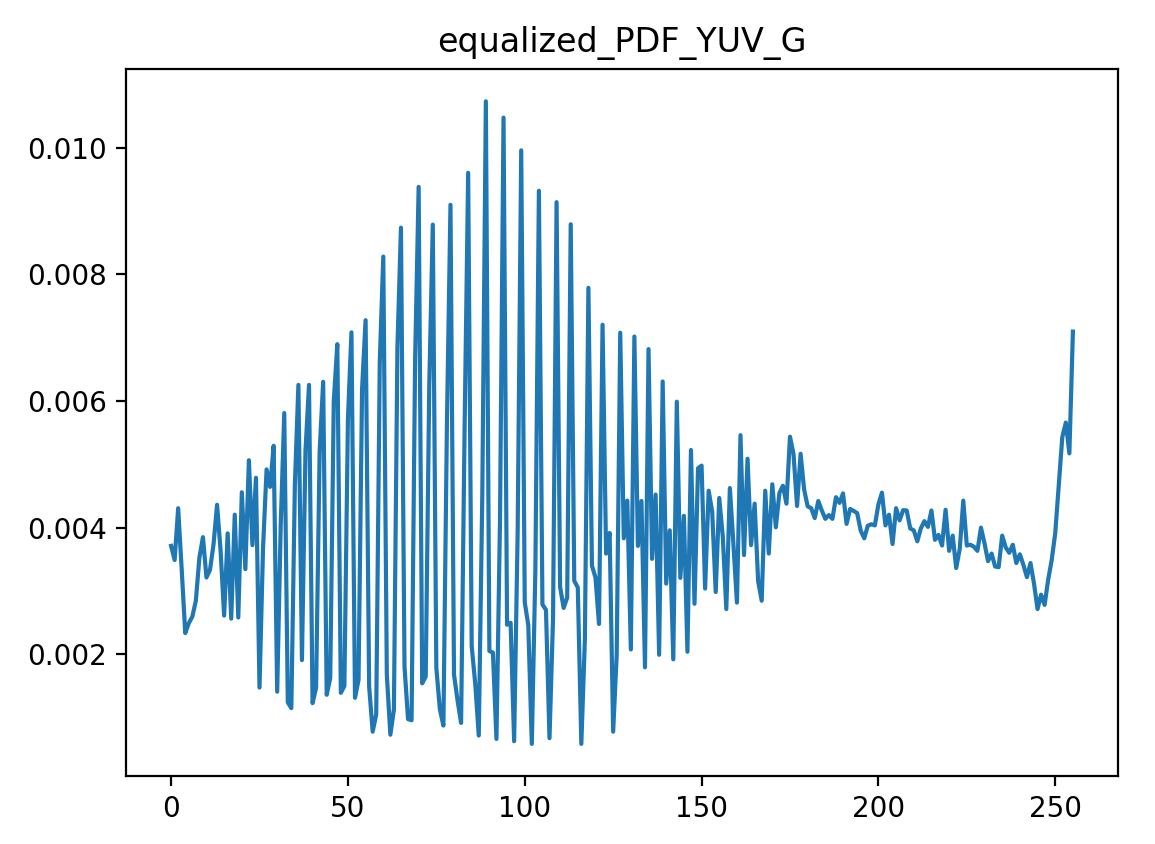
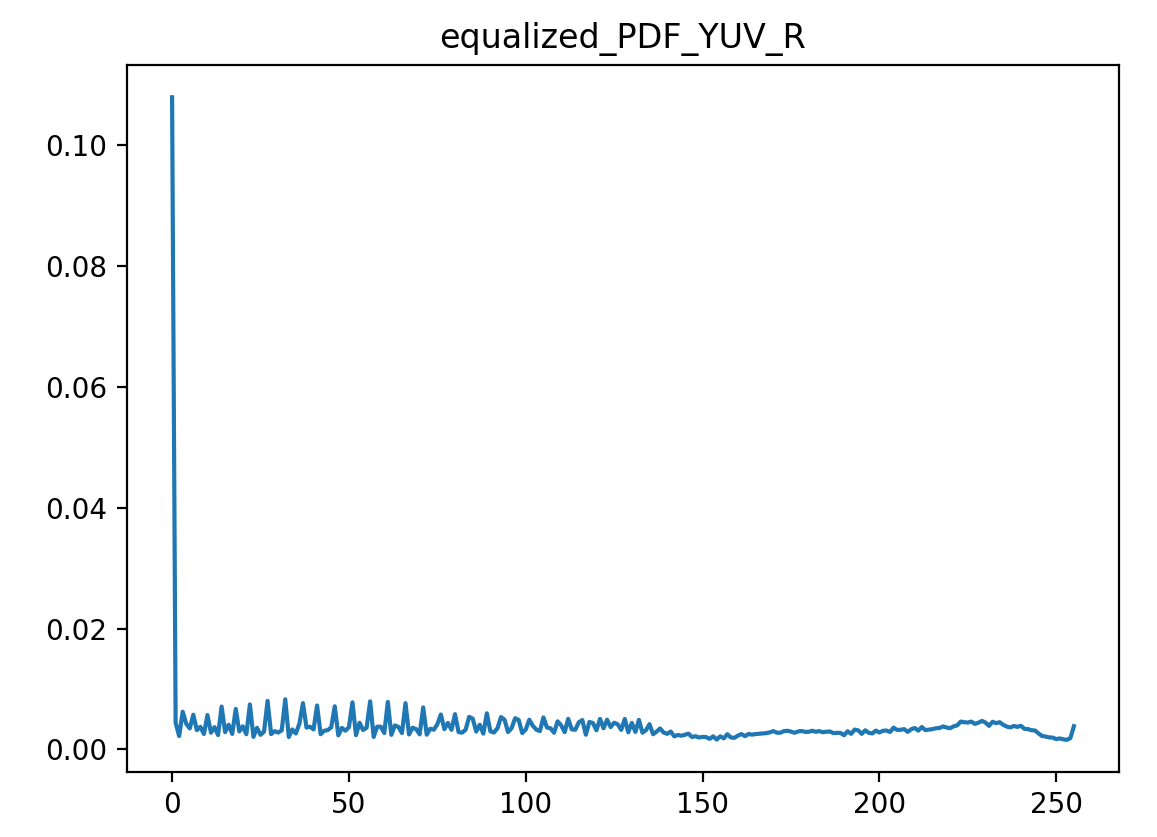
**B. Result**

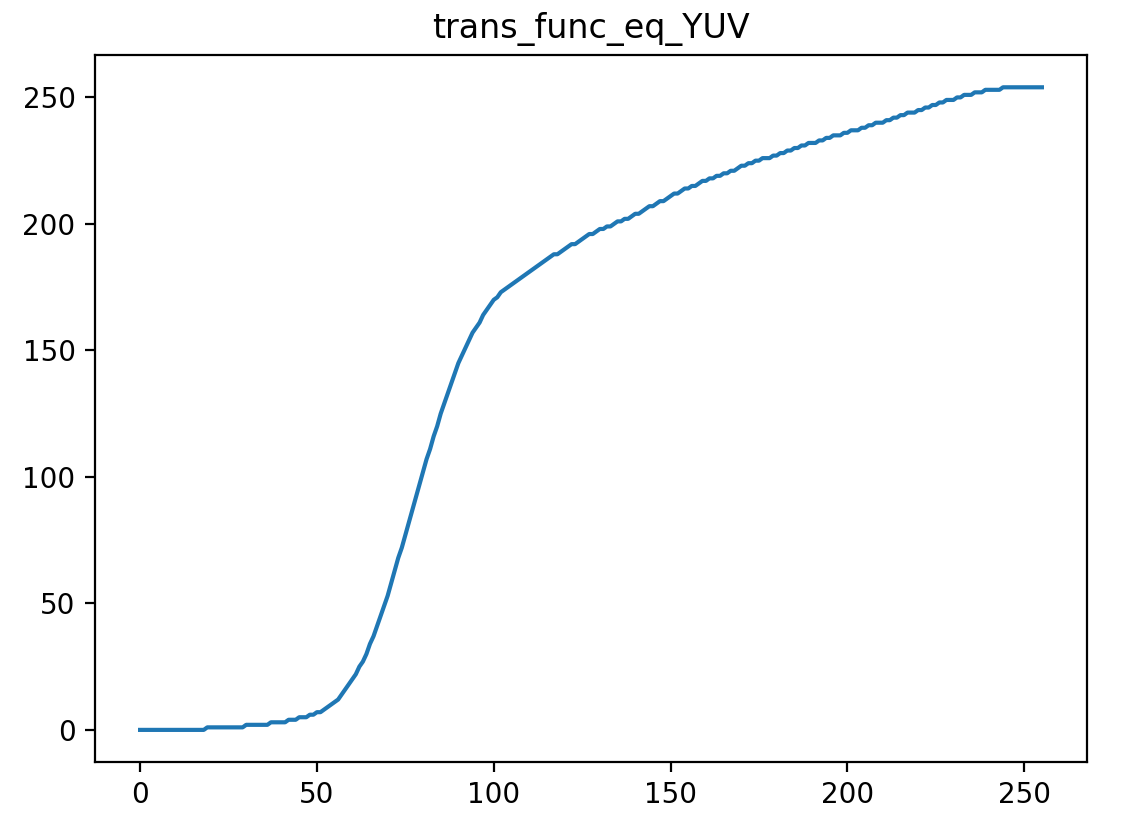
-Original Image / Histogram

3-2)-B와 동일

-Output image / Histogram







**4. Histogram Matching**

**-**ref.jpg를 reference image로 사용함

**1) Grayscale Image**

**A. Code**

#include "hist\_func.h"

#include <iostream>

using namespace std;

void hist\_eq(Mat &input, Mat &equalized, G \*trans\_func, float \*CDF);

void func\_inverse(G \*trans\_func, G \*inverse\_func);

int main() {

/\*

Input: Original Image, Reference Image

Output:

-histogram-matched image

-1 transfer function(txt file)

-1 histogram of the original image(txt file)

-1 histogram of the ouput image(txt file)

\*/

Mat input = imread("input.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat input\_gray;

Mat ref = imread("ref.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat ref\_gray;

cvtColor(input, input\_gray, CV\_RGB2GRAY); // Convert RGB to Gray

cvtColor(ref, ref\_gray, CV\_RGB2GRAY); // Convert RGB to Gray

Mat equalized = input\_gray.clone();

Mat equalized\_ref = ref\_gray.clone();

Mat matched = input\_gray.clone();

//PDF or Transfer function txt

FILE \*f\_PDF;

//FILE \*f\_ref\_PDF\_gray;

FILE \*f\_matched\_PDF\_gray;

FILE \*f\_trans\_func\_match\_gray;

fopen\_s(&f\_PDF, "PDF.txt", "w+");

//fopen\_s(&f\_ref\_PDF\_gray, "ref\_PDF.txt", "w+");

fopen\_s(&f\_matched\_PDF\_gray, "matched\_PDF\_gray.txt", "w+");

fopen\_s(&f\_trans\_func\_match\_gray, "trans\_func\_match\_gray.txt", "w+");

//1. Compute the transfer functions s = T(r) for histogram equalization of an input image, and perform histogram equalization

float \*PDF\_input = cal\_PDF(input\_gray); // PDF of Input image(Grayscale) : [L]

float \*CDF\_input = cal\_CDF(input\_gray); // CDF of Input image(Grayscale) : [L]

G Tr[L] = { 0 }; // transfer function

hist\_eq(input\_gray, equalized, Tr, CDF\_input); // histogram equalization on grayscale image

//2. Compute the transfer functions s = G(z) for histogram equalization of the specified PDF

float \*PDF\_ref = cal\_PDF(ref\_gray); // equalized PDF (grayscale)

float \*CDF\_ref = cal\_CDF(ref\_gray);

G Gz[L] = { 0 }; // transfer function

hist\_eq(ref\_gray, equalized\_ref, Gz, CDF\_ref);

float \*z = cal\_PDF(equalized\_ref);

//3. Apply the intensity mapping from r to z

G inverseGz[L] = { 0 };

func\_inverse(Gz, inverseGz);

// perform the transfer function

for (int i = 0; i < input.rows; i++)

for (int j = 0; j < input.cols; j++)

matched.at<G>(i, j) = inverseGz[Tr[input\_gray.at<G>(i, j)]];

float \*matched\_PDF\_gray = cal\_PDF(matched);

//write files

for (int i = 0; i < L; i++) {

// write PDF

fprintf(f\_PDF, "%d\t%f\n", i, PDF\_input[i]);

//fprintf(f\_ref\_PDF\_gray, "%d\t%f\n", i, PDF\_ref[i]);

fprintf(f\_matched\_PDF\_gray, "%d\t%f\n", i, matched\_PDF\_gray[i]);

// write transfer functions

fprintf(f\_trans\_func\_match\_gray, "%d\t%d\n", i, inverseGz[Tr[i]]);

}

// memory release

fclose(f\_PDF);

//fclose(f\_ref\_PDF);

fclose(f\_matched\_PDF\_gray);

fclose(f\_trans\_func\_match\_gray);

////////////////////// Show each image ///////////////////////

namedWindow("Grayscale", WINDOW\_AUTOSIZE);

imshow("Grayscale", input\_gray);

namedWindow("Matched", WINDOW\_AUTOSIZE);

imshow("Matched", matched);

//////////////////////////////////////////////////////////////

waitKey(0);

return 0;

}

// histogram equalization

void hist\_eq(Mat &input, Mat &equalized, G \*trans\_func, float \*CDF) {

// compute transfer function

for (int i = 0; i < L; i++) {

trans\_func[i] = (G)((L - 1) \* CDF[i]);

}

// perform the transfer function

for (int i = 0; i < input.rows; i++)

for (int j = 0; j < input.cols; j++)

equalized.at<G>(i, j) = trans\_func[input.at<G>(i, j)];

}

//get inverse function G-1

void func\_inverse(G \*trans\_func, G \*inverse\_func){

/\* Return z = G-1(s) \*/

bool checked[L] = { 0 };

for (int i = 0; i < L; i++) {

if (checked[trans\_func[i]])

continue;

inverse\_func[trans\_func[i]] = (G)i;

checked[trans\_func[i]] = 1;

}

for (int i = 0; i < L; i++)

if (!checked[i])

inverse\_func[i] = (G)inverse\_func[i - 1];

}

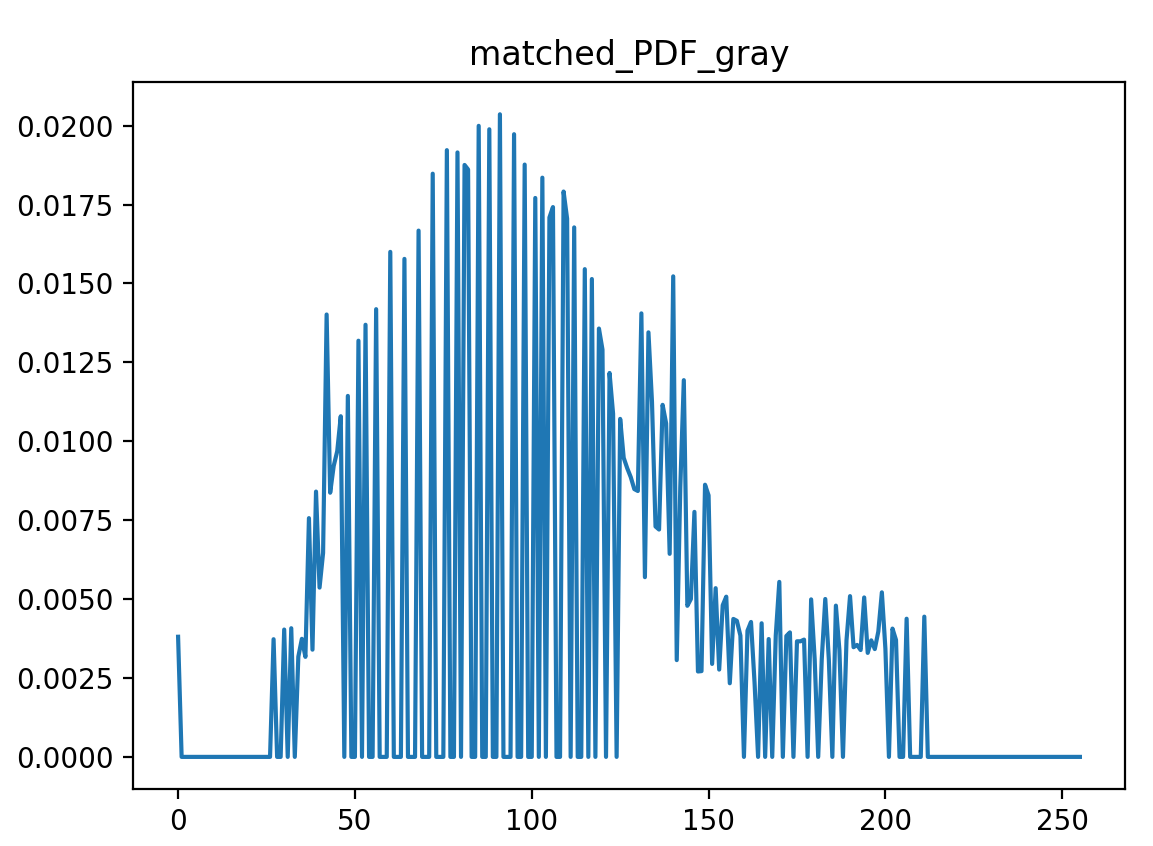
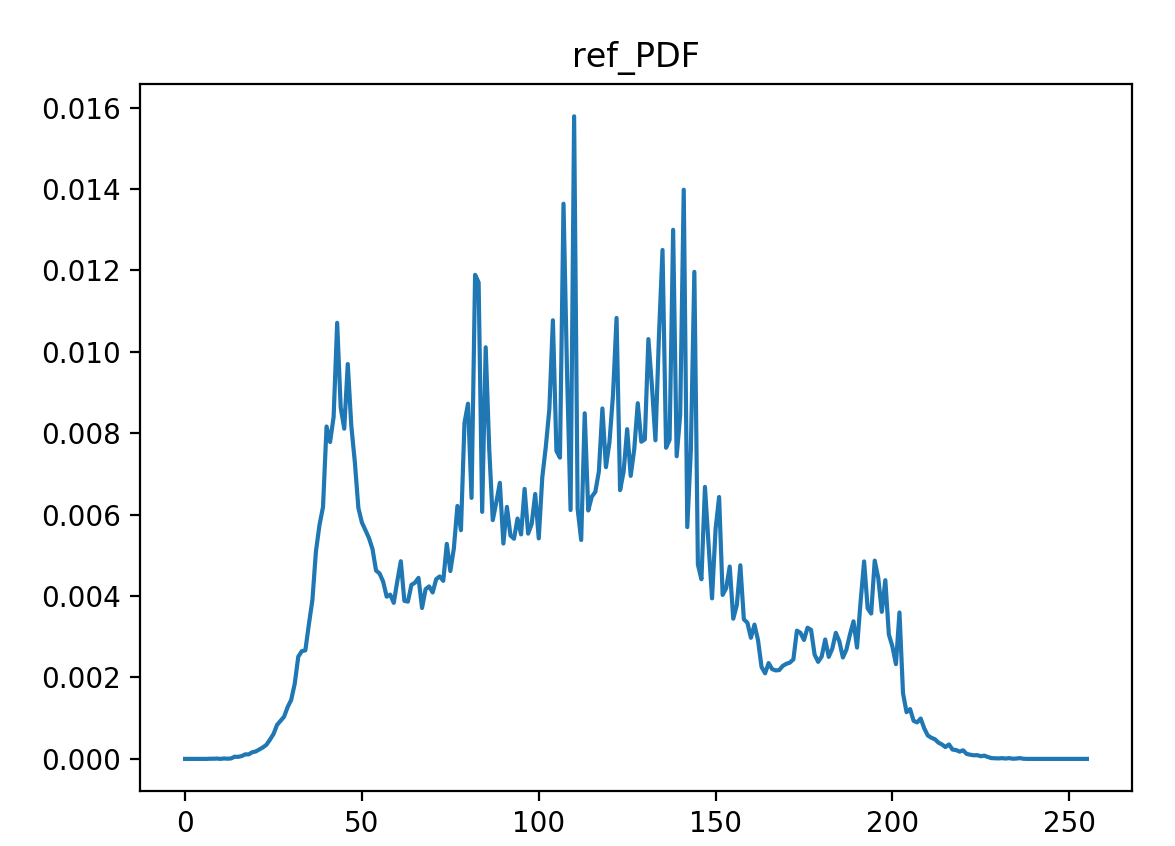
**B. Result**

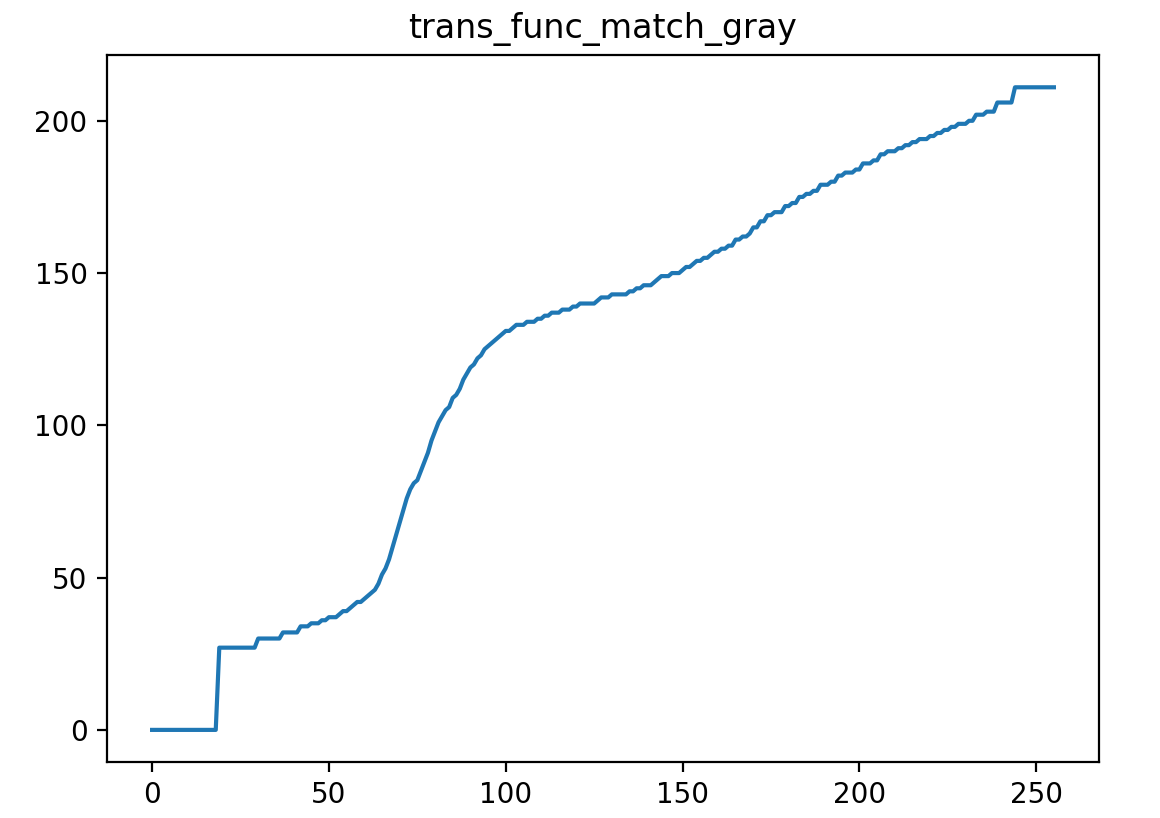
**-Original Image / Histogram**

1-B 와 동일

**-Output image / Histogram**







**2) RGB**

**A. Code**

#include "hist\_func.h"

#include <iostream>

using namespace std;

void hist\_eq(Mat &input, Mat &equalized, G \*trans\_func, float \*CDF);

void func\_inverse(G \*trans\_func, G \*inverse\_func);

int main() {

/\*

Input: Original Image, Reference Image

Output:

-histogram-matched image

-1 transfer function(txt file)

-1 histogram of the original image(txt file)

-1 histogram of the ouput image(txt file)

\*/

Mat input = imread("input.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat input\_matched\_YUV;

Mat ref = imread("ref.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat ref\_matched\_YUV;

//1. Convert the color image RGB to YUV

cvtColor(input, input\_matched\_YUV, CV\_RGB2YUV);

cvtColor(ref, ref\_matched\_YUV, CV\_RGB2YUV);

//Split each channel(Y, U, V);

Mat input\_channels[3]; Mat ref\_channels[3];

split(input\_matched\_YUV, input\_channels); split(ref\_matched\_YUV, ref\_channels);

Mat input\_Y = input\_channels[0]; Mat ref\_Y = ref\_channels[0];

//PDF or transfer function txt files

FILE \*f\_PDF\_R, \*f\_PDF\_G, \*f\_PDF\_B;

FILE \*f\_matched\_PDF\_R, \*f\_matched\_PDF\_G, \*f\_matched\_PDF\_B;

FILE \*f\_trans\_func\_match\_YUV;

fopen\_s(&f\_PDF\_R, "PDF\_R(match).txt", "w+"); fopen\_s(&f\_PDF\_G, "PDF\_G(match).txt", "w+"); fopen\_s(&f\_PDF\_B, "PDF\_B(match).txt", "w+");

fopen\_s(&f\_matched\_PDF\_R, "matched\_PDF\_R.txt", "w+"); fopen\_s(&f\_matched\_PDF\_G, "matched\_PDF\_G.txt", "w+"); fopen\_s(&f\_matched\_PDF\_B, "matched\_PDF\_B.txt", "w+");

fopen\_s(&f\_trans\_func\_match\_YUV, "trans\_func\_match\_YUV.txt", "w+");

// 2. Apply the histogram matching for Y channel only

// Compute the transfer functions s = T(r) for histogram equalization of input's Y channel, and perform histogram equalization

float \*\*PDF\_input\_RGB = cal\_PDF\_RGB(input);

float \*CDF\_input\_YUV = cal\_CDF(input\_Y);

G Tr\_YUV[L] = { 0 }; //transfer function s = T(r)

hist\_eq(input\_Y, input\_Y, Tr\_YUV, CDF\_input\_YUV);

// Compute the transfer functions s = G(z) for histogram equalization of ref's Y channel

//float \*\*PDF\_ref\_RGB = cal\_PDF\_RGB(ref);

float \*CDF\_ref\_YUV = cal\_CDF(ref\_Y);

G Gz\_YUV[L] = { 0 }; //transfer function s = T(r)

hist\_eq(ref\_Y, ref\_Y, Gz\_YUV, CDF\_ref\_YUV);

// Apply the intensity mapping from r to z

G inverseGz[L] = { 0 };

func\_inverse(Gz\_YUV, inverseGz);

// perform the transfer function

for (int i = 0; i < input.rows; i++)

for (int j = 0; j < input.cols; j++)

input\_Y.at<G>(i, j) = inverseGz[Tr\_YUV[input\_Y.at<G>(i, j)]];

//3. Convert the histogram-matched Y and UV to the color image(RGB)

// merge Y. U, V channels

merge(input\_channels, 3, input\_matched\_YUV);

// YUV -> RGB

cvtColor(input\_matched\_YUV, input\_matched\_YUV, CV\_YUV2RGB);

// output PDF

float \*\*PDF\_output\_RGB = cal\_PDF\_RGB(input\_matched\_YUV);

//write files

for (int i = 0; i < L; i++) {

// write PDF of original image

fprintf(f\_PDF\_R, "%d\t%f\n", i, PDF\_input\_RGB[i][0]); fprintf(f\_PDF\_G, "%d\t%f\n", i, PDF\_input\_RGB[i][1]); fprintf(f\_PDF\_B, "%d\t%f\n", i, PDF\_input\_RGB[i][2]);

// write PDF of output image

fprintf(f\_matched\_PDF\_R, "%d\t%f\n", i, PDF\_output\_RGB[i][0]); fprintf(f\_matched\_PDF\_G, "%d\t%f\n", i, PDF\_output\_RGB[i][1]); fprintf(f\_matched\_PDF\_B, "%d\t%f\n", i, PDF\_output\_RGB[i][2]);

// write transfer functions

fprintf(f\_trans\_func\_match\_YUV, "%d\t%d\n", i, inverseGz[Tr\_YUV[i]]);

}

//file close

fclose(f\_PDF\_R); fclose(f\_PDF\_G); fclose(f\_PDF\_B);

fclose(f\_matched\_PDF\_R); fclose(f\_matched\_PDF\_G); fclose(f\_matched\_PDF\_B);

fclose(f\_trans\_func\_match\_YUV);

////////////////////// Show each image ///////////////////////

namedWindow("RGB", WINDOW\_AUTOSIZE);

imshow("RGB", input);

namedWindow("Equalized\_YUV", WINDOW\_AUTOSIZE);

imshow("Equalized\_YUV", input\_matched\_YUV);

//////////////////////////////////////////////////////////////

waitKey(0);

return 0;

}

// histogram equalization

void hist\_eq(Mat &input, Mat &equalized, G \*trans\_func, float \*CDF) {

// compute transfer function

for (int i = 0; i < L; i++)

trans\_func[i] = (G)((L - 1) \* CDF[i]);

// perform the transfer function

for (int i = 0; i < input.rows; i++)

for (int j = 0; j < input.cols; j++)

equalized.at<G>(i, j) = trans\_func[input.at<G>(i, j)];

}

//get inverse function G-1

void func\_inverse(G \*trans\_func, G \*inverse\_func) {

/\* Return z = G-1(s) \*/

bool checked[L] = { 0 };

for (int i = 0; i < L; i++) {

if (checked[trans\_func[i]])

continue;

inverse\_func[trans\_func[i]] = (G)i;

checked[trans\_func[i]] = 1;

}

for (int i = 0; i < L; i++)

if (!checked[i])

inverse\_func[i] = (G)inverse\_func[i - 1];

}

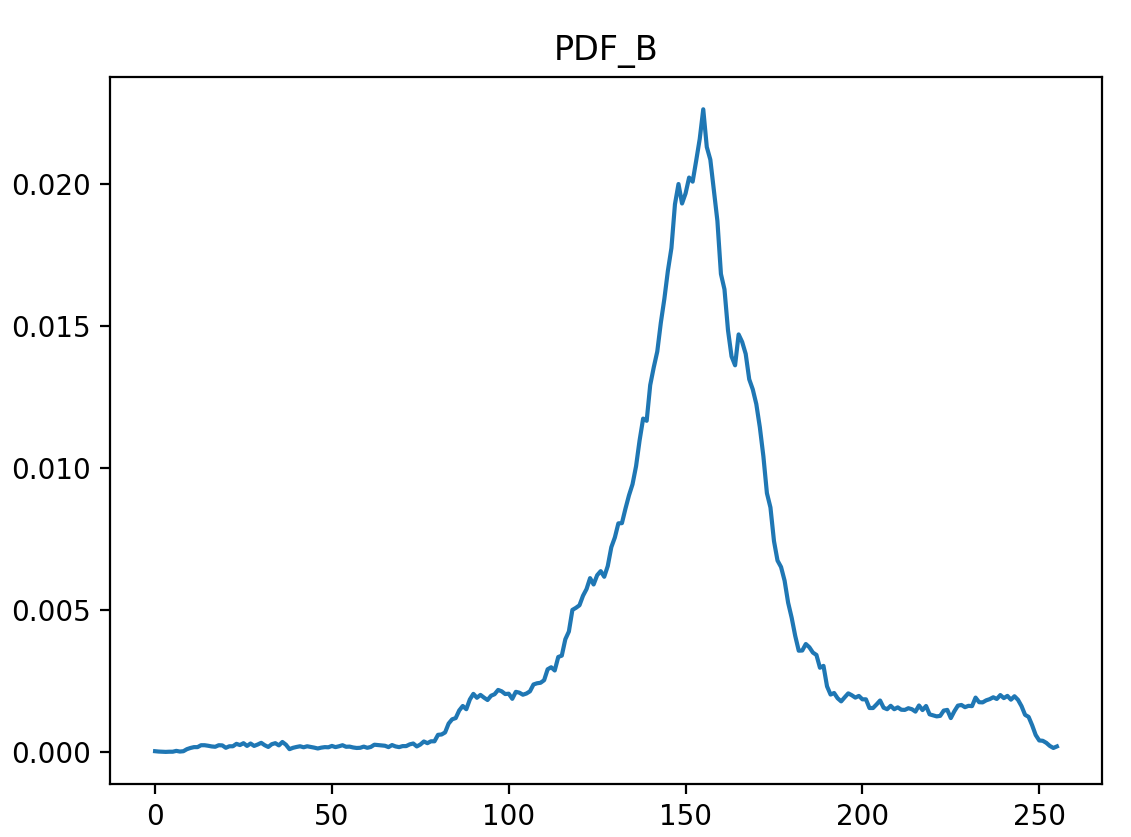
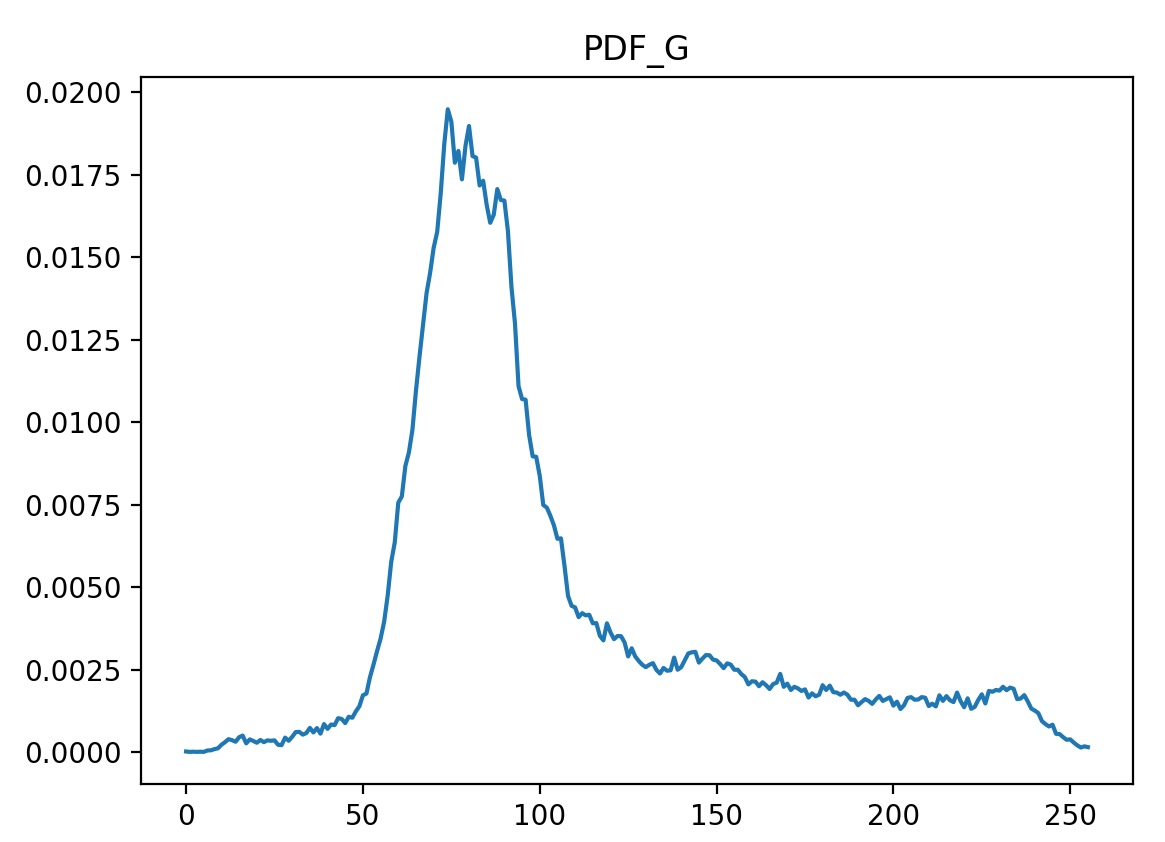
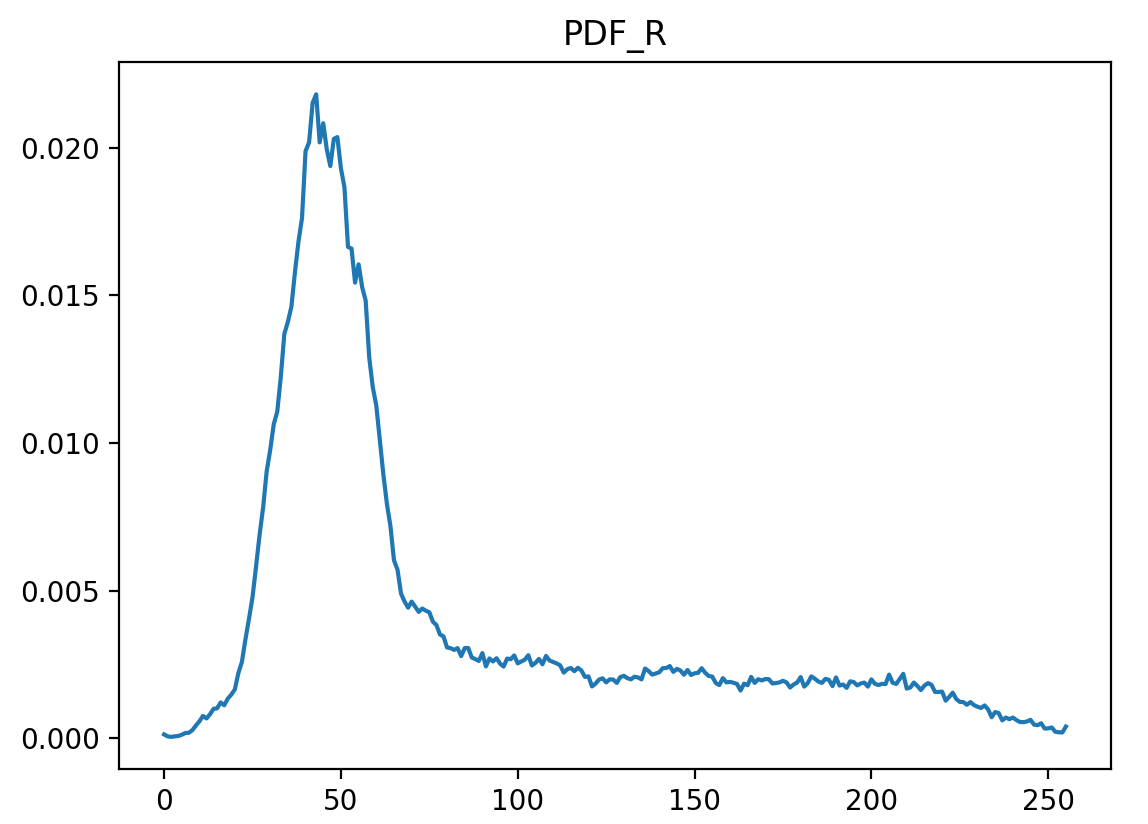
**B. Result**

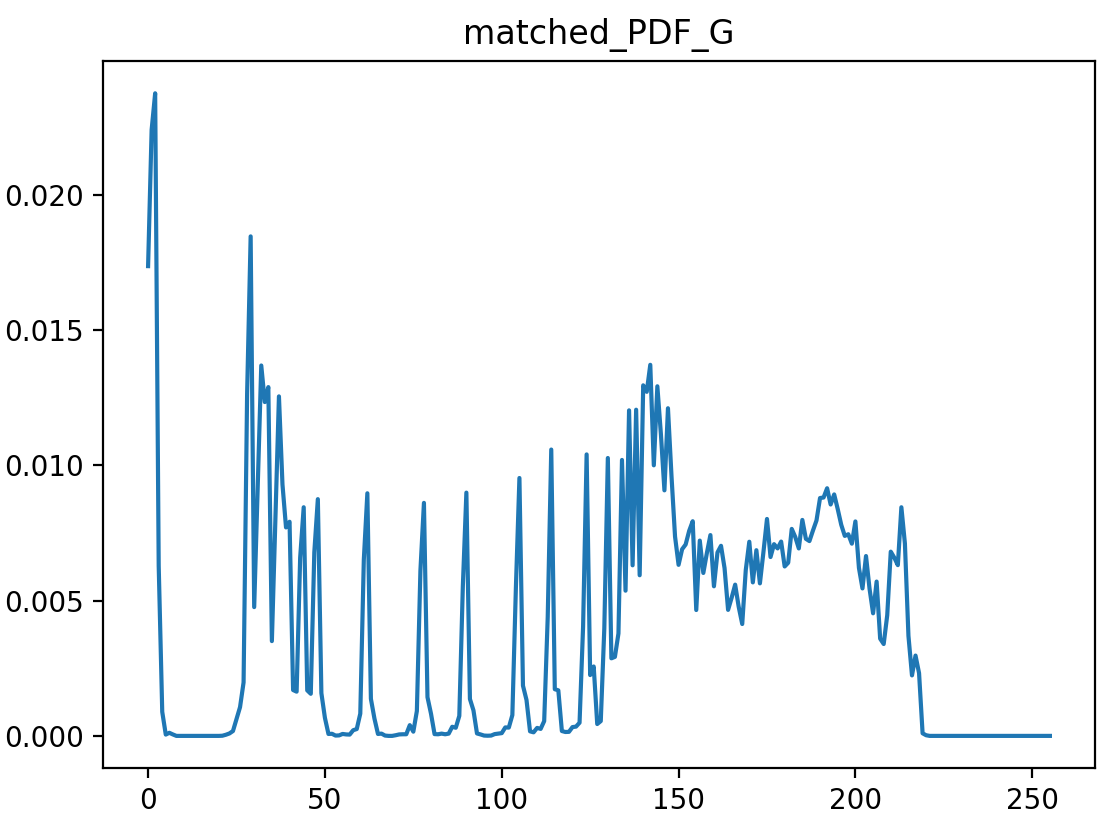
-Original Image / Histogram

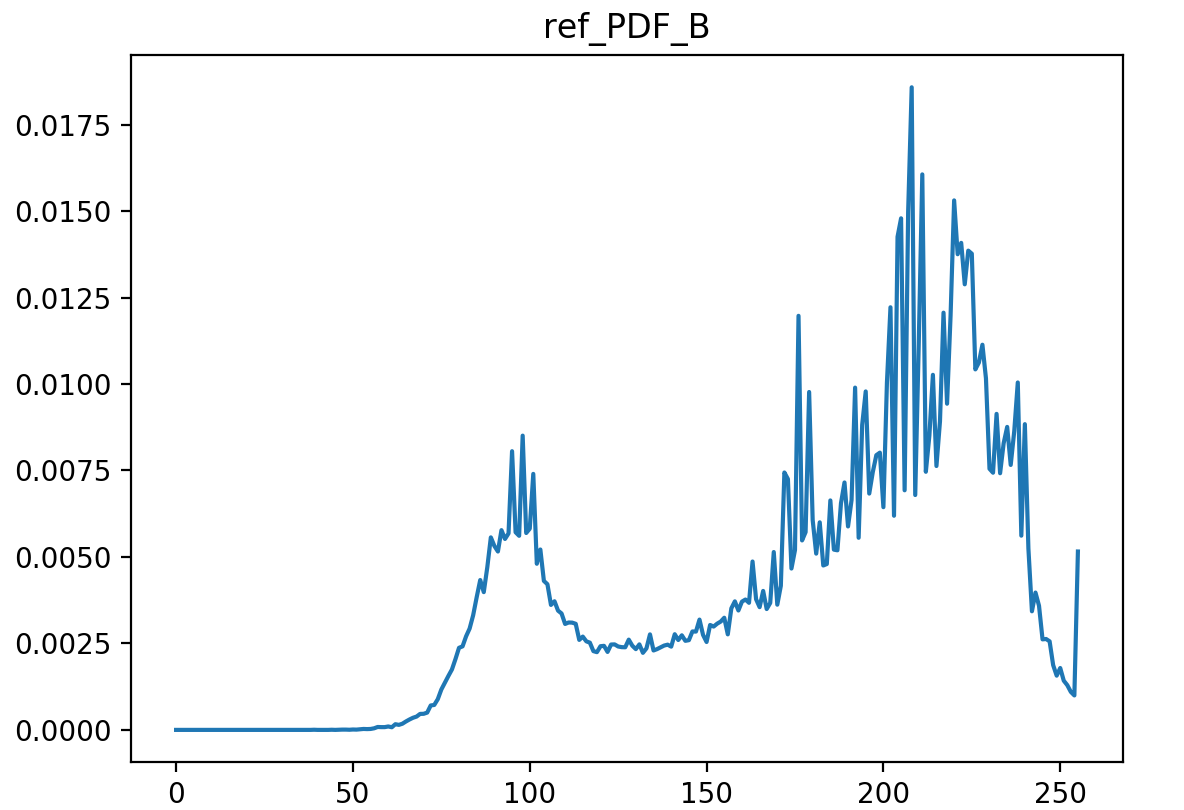
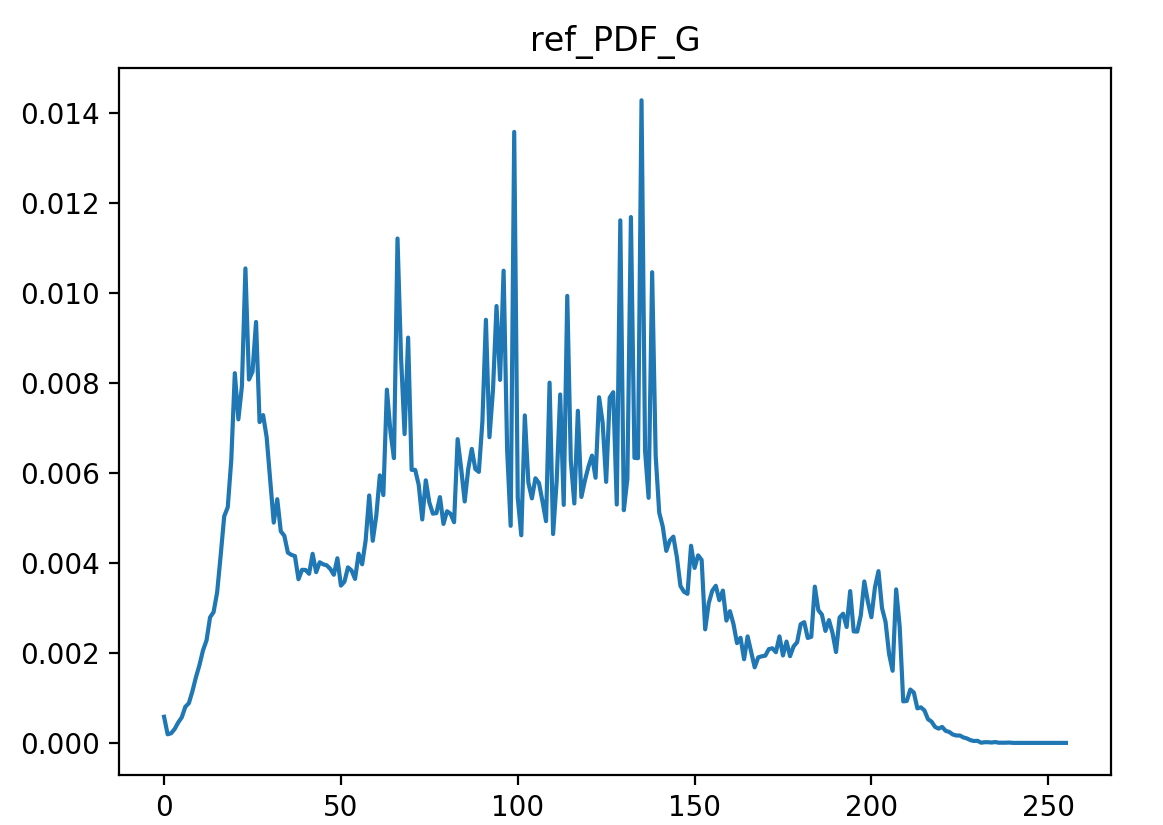
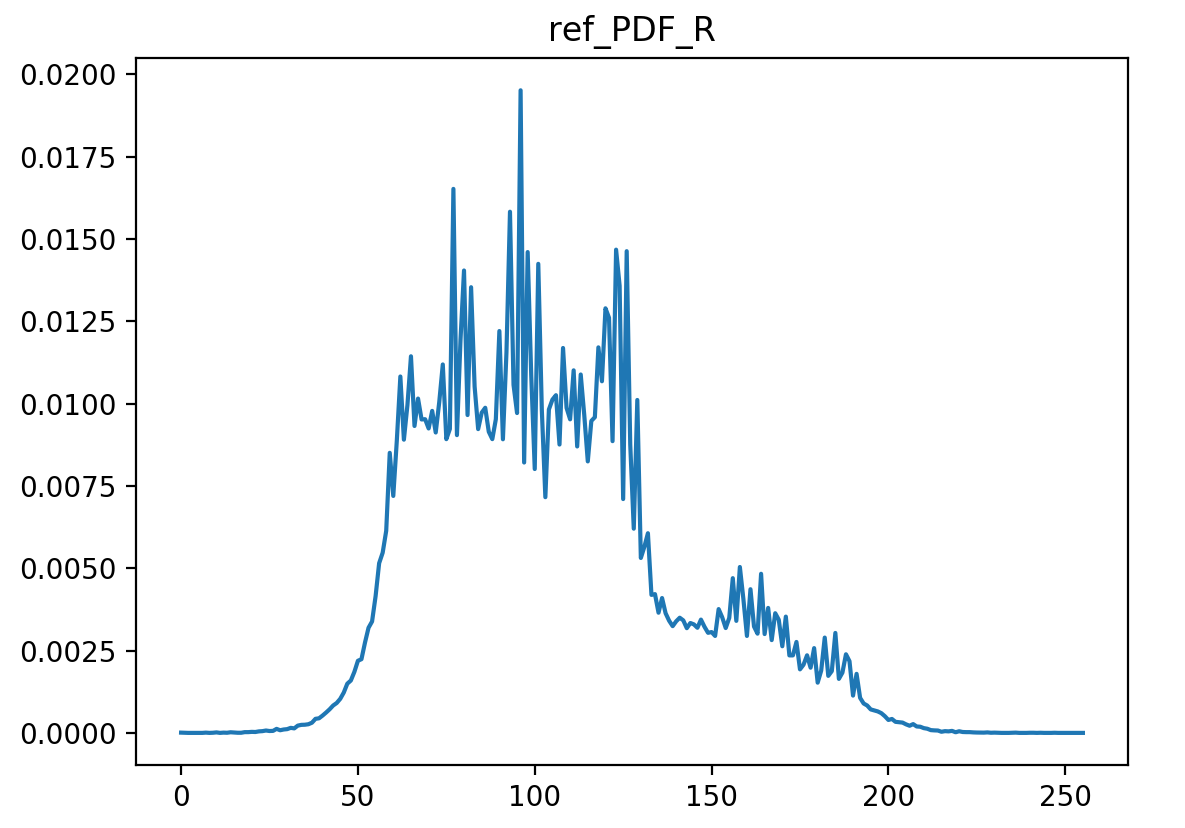
3-2)-B와 동일

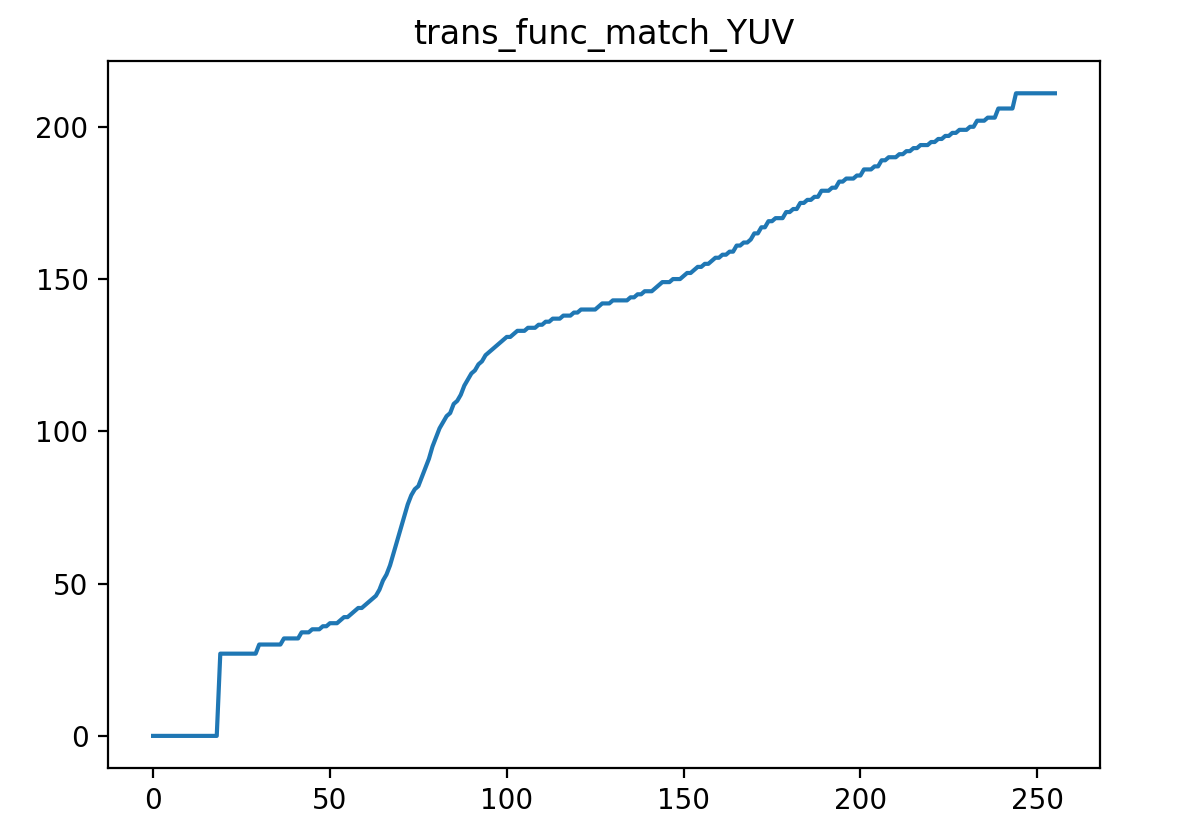
-Output image / Histogram











**\*Code for drawing plot**

import matplotlib.pyplot as plt  
  
def getPlot(dir):  
 x\_axis = [i for i in range(256)]  
 y\_axis = []  
 file = open(dir, 'r')  
 while True:  
 line = file.readline()  
 if not line:  
 break  
 \_, y = map(float, line.split())  
 y\_axis.append(y)  
  
 file.close()  
  
 plt.plot(x\_axis, y\_axis)  
 plt.title(dir[:-4])  
 #plt.legend()  
 plt.show()