Lab 1: Image Filtering

Course Number and Title: ECE 63700 Image Processing I (Spring 2022)

Instructor: Prof. Charles A. Bouman

Author: **Zhankun Luo**

3. FIR Low Pass Filter

3.1. expression for low pass filter $H\left(e^{j\mu}, e^{j\nu}\right)$

let $h(m,n)$ be a low pass filter

$$h(m,n) = \begin{cases} 1/81 & \text{for } |m| \leq 4 \text{ and } |n| \leq 4 \\ 0 & \text{otherwise} \end{cases}$$

solution

$$h(m,n) = h_1(m)h_2(n) = \left(\frac{1}{9}\sum_{k=-4}^{4}\delta(m-k)\right)\left(\frac{1}{9}\sum_{l=-4}^{4}\delta(n-l)\right)$$

$$H\left(e^{j\mu}, e^{j\nu}\right) \equiv \left(\sum_{m=-\infty}^{\infty} h_1(m)e^{-jm\mu}\right)\left(\sum_{n=-\infty}^{\infty} h_2(n)e^{-jn\nu}\right)$$

Notice that

$$\left(\sum_{m=-\infty}^{\infty} h_1(m)e^{-jm\mu}\right) = \frac{1}{9}\left(\sum_{k=-4}^{4} e^{-jk\mu}\right) = \frac{e^{j4\mu}}{9}\frac{1-e^{-j9\mu}}{1-e^{-j\mu}} = \frac{\sin\frac{9\mu}{2}}{9\sin\frac{\mu}{2}}$$

Similarly, we can compute $\left(\sum_{n=-\infty}^{\infty} h_2(n)e^{-jn\nu}\right) = \frac{\sin\frac{9\nu}{2}}{9\sin\frac{\nu}{2}}$

$$H\left(e^{j\mu}, e^{j\nu}\right) = \frac{\sin\frac{9\mu}{2}}{9\sin\frac{\mu}{2}}\frac{\sin\frac{9\nu}{2}}{9\sin\frac{\nu}{2}}$$

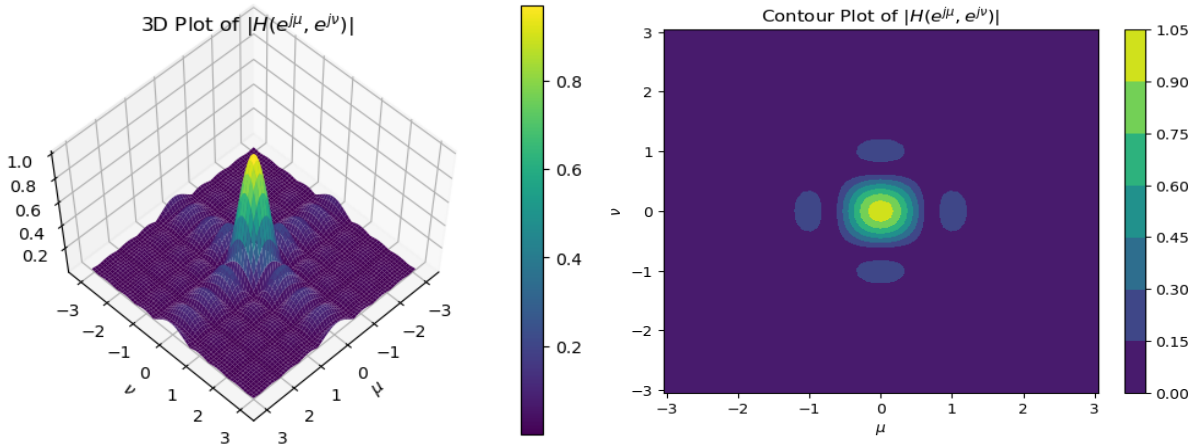
3.2. plot for amplitude of low pass filter $|H\left(e^{j\mu}, e^{j\nu}\right)|$

solution

Run visualization Python script **vis_3.py** with

```
python ./visualize/vis_3.py
```

The left figure below is a 3D mesh plot for $|H\left(e^{j\mu}, e^{j\nu}\right)|$, the right contour figure is a contour plot for $|H\left(e^{j\mu}, e^{j\nu}\right)|$



3D Mesh(left) and Contour(right) Plots for Amplitude of Low Pass Filter

3.3. input color image `img03.tif` and 3.4. output filtered color image

solution

Run the executable program **soln_3** compiled by **soln_3.c** with

```
./soln_3 img03.tif
```

The input color image `img03.tif` and the corresponding output filtered color image are shown below

Input Color Image **img03.tif***(left) and Output Filtered Color Image(right)*

3.5. listing of C codes

solution

The corresponding function `filter_FIR_lowpass()` for section 3 in **filter.c**

```
void filter_FIR_lowpass(double **a, double**a_t,
                        int16_t W, int16_t H,
                        int16_t kernel_size) {
    // assert (kernel_size >= 1 &&
    // kernel_size%2 == 1 &&
    // kernel_size <= W && kernel_size <= H)
    int16_t width = kernel_size / 2;
    double coef_sum = 1.0 / (kernel_size*kernel_size);
    /* allocate memory and set 0 */
    double **a_m = (double **)get_img(W, H, sizeof(double));
    for (int16_t i=0; i < H; i++) {
        memset(a_m[i], 0, W*sizeof(double));
        memset(a_t[i], 0, W*sizeof(double));
    }
    for (int16_t i = 0; i < H; i++) {
        for (int16_t dj = -width; dj <= width; dj++) {
            for (int16_t j = width; j < W-width; j++) {
                a_m[i][j] += a[i][j+dj];
            }
        }
    }
    for (int16_t di = -width; di <= width; di++) {
        for (int16_t i = width; i < H-width; i++) {
            for (int16_t j = width; j < W-width; j++) {
                a_t[i][j] += a_m[i+di][j];
            }
        }
    }
    for (int16_t i = width; i < H-width; i++) {
        for (int16_t j = width; j < W-width; j++) {
            a_t[i][j] *= coef_sum;
        }
    }
    /* fill the border with pixels of input array */
    if (width > 0) {
```

```
        for (int16_t i = 0; i < width; i++) {
            for (int16_t j = 0; j < W; j++) {
                a_t[i][j] = a[i][j];
            }
        }
        for (int16_t i = H-width; i < H; i++) {
            for (int16_t j = 0; j < W; j++) {
                a_t[i][j] = a[i][j];
            }
        }
        for (int16_t i = width; i < H-width; i++) {
            for (int16_t j = 0; j < width; j++) {
                a_t[i][j] = a[i][j];
            }
        }
        for (int16_t i = width; i < H-width; i++) {
            for (int16_t j = W-width; j < W; j++) {
                a_t[i][j] = a[i][j];
            }
        }
    }
    free_img( (void**)a_m );
}
```

4. FIR Sharpening Filter

4.1. expression for low pass filter $H\left(e^{j\mu}, e^{j\nu}\right)$

let $h(m,n)$ be a low pass filter

$$h(m,n) = \begin{cases} 1/25 & \text{for } |m| \leq 2 \text{ and } |n| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

solution

$$h(m,n) = h_1(m)h_2(n) = \left(\frac{1}{5}\sum_{k=-2}^{2}\delta(m-k)\right)\left(\frac{1}{5}\sum_{l=-2}^{2}\delta(n-l)\right)$$

$$H\left(e^{j\mu}, e^{j\nu}\right) \equiv \left(\sum_{m=-\infty}^{\infty} h_1(m)e^{-jm\mu}\right)\left(\sum_{n=-\infty}^{\infty} h_2(n)e^{-jn\nu}\right)$$

Notice that

$$\left(\sum_{m=-\infty}^{\infty} h_1(m)e^{-jm\mu}\right) = \frac{1}{5}\left(\sum_{k=-2}^{2} e^{-jk\mu}\right) = \frac{e^{j4\mu}}{5}\frac{1-e^{-j5\mu}}{1-e^{-j\mu}} = \frac{\sin\frac{5\mu}{2}}{5\sin\frac{\mu}{2}}$$

Similarly, we can compute $\left(\sum_{n=-\infty}^{\infty} h_2(n)e^{-jn\nu}\right) = \frac{\sin\frac{5\nu}{2}}{5\sin\frac{\nu}{2}}$

$$H\left(e^{j\mu}, e^{j\nu}\right) = \frac{\sin\frac{5\mu}{2}}{5\sin\frac{\mu}{2}}\frac{\sin\frac{5\nu}{2}}{5\sin\frac{\nu}{2}}$$

4.2. expression for unsharp mask filter $G\left(e^{j\mu}, e^{j\nu}\right)$

The unsharp mask filter is then given by, where $\lambda > 0$ is a constant

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n))$$

solution

$$G\left(e^{j\mu}, e^{j\nu}\right) = (1+\lambda)\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}\delta(m,n)e^{-j(m\mu+n\nu)} - \lambda H\left(e^{j\mu}, e^{j\nu}\right)$$

Notice $\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}\delta(m,n)e^{-j(m\mu+n\nu)} = 1$

$$G\left(e^{j\mu}, e^{j\nu}\right) = (1+\lambda) - \lambda\frac{\sin\frac{5\mu}{2}}{5\sin\frac{\mu}{2}}\frac{\sin\frac{5\nu}{2}}{5\sin\frac{\nu}{2}}$$

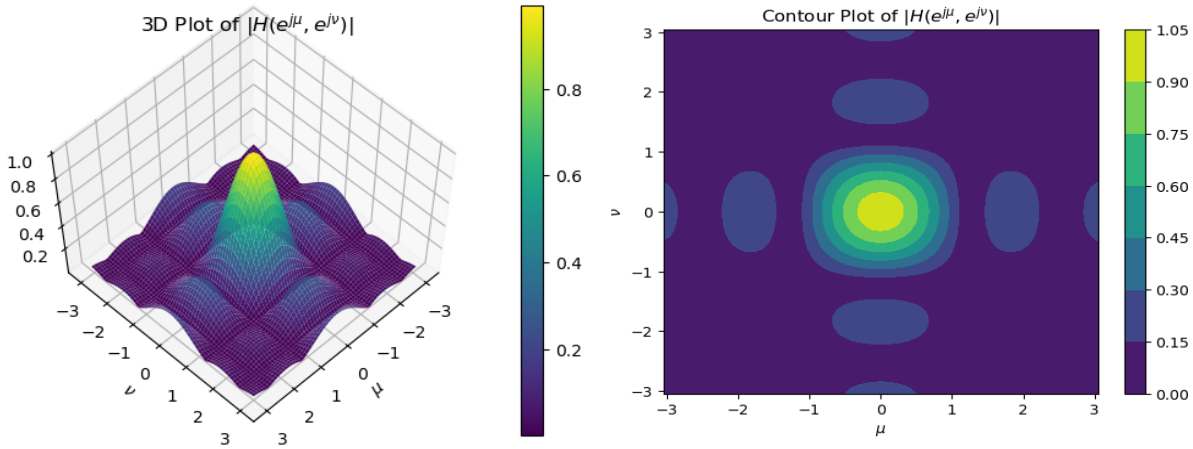
4.3. plot of amplitude for low pass filter $|H\left(e^{j\mu}, e^{j\nu}\right)|$

solution

Run visualization Python script **vis_4.py** with

```
python ./visualize/vis_4.py
```

The left figure below is a 3D mesh plot for $|H\left(e^{j\mu}, e^{j\nu}\right)|$, the right contour figure is a contour plot for $|H\left(e^{j\mu}, e^{j\nu}\right)|$



3D Mesh(left) and Contour(right) Plots for Amplitude of Low Pass Filter

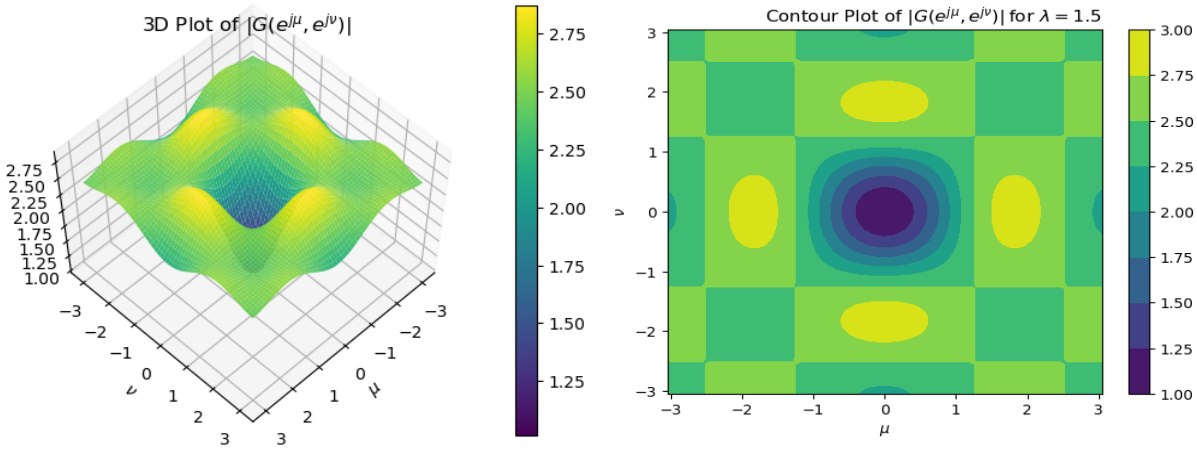
4.4. plot of amplitude for unsharp mask filter $|G\left(e^{j\mu}, e^{j\nu}\right)|$ when $\lambda = 1.5$

solution

Run visualization Python script **vis_4.py** with

```
python ./visualize/vis_4.py
```

The left figure below is a 3D mesh plot for $|G\left(e^{j\mu}, e^{j\nu}\right)|$, the right contour figure is a contour plot for $|G\left(e^{j\mu}, e^{j\nu}\right)|$ when $\lambda = 1.5$



3D Mesh(left) and Contour(right) Plots for Amplitude of Unsharp Mask Filter

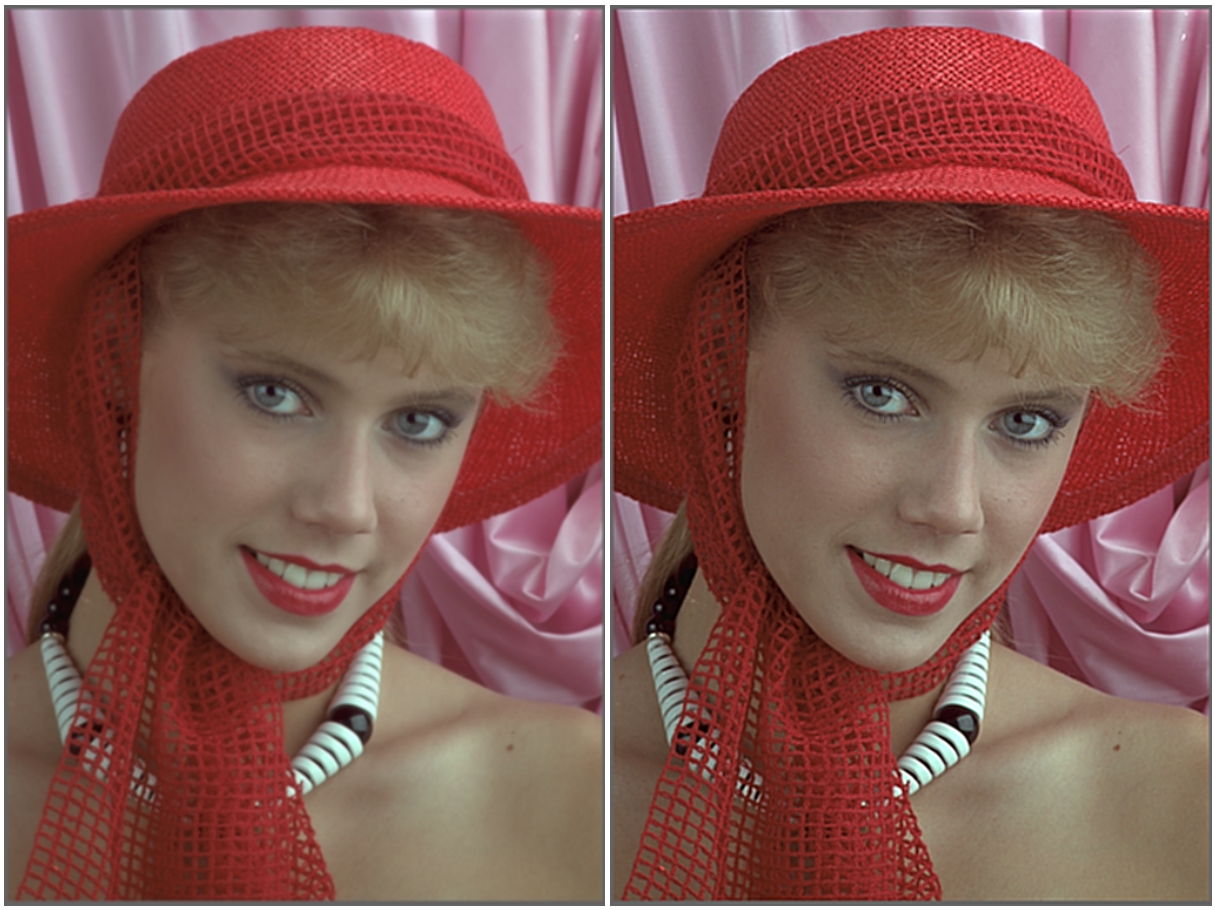
4.5. input color image `imgblur.tif` 4.6. output sharpened color image for $\lambda = 1.5$

solution

Run executable program **soln_4** compiled by **soln_4.c** with arguments λ = 1.5, input image `imgblur.tif`

```
./soln_4 1.5 imgblur.tif
```

The left image is the input color image `imgblur.tif` and the right image is the output sharpened color image



Input Color Image **imgblur.tif***(left) and Output Sharpened Color Image(right)*

4.7. listing of C codes

solution

The corresponding function `filter_FIR_sharpen()` for section 4 in **filter.c**

```
void filter_FIR_sharpen(double **a, double**a_t,
                        int16_t W, int16_t H,
                        int16_t kernel_size, double lambda) {
    filter_FIR_lowpass(a, a_t, W, H, kernel_size);
    int16_t width = kernel_size / 2;
    for (int16_t i = width; i < H-width; i++) {
        for (int16_t j = width; j < W-width; j++) {
            a_t[i][j] *= (-lambda);
            a_t[i][j] += (1+lambda) * a[i][j];
        }
    }
}
```

5. IIR Filter

5.1. expression for IIR filter $H\left(e^{j\mu}, e^{j\nu}\right)$

let $h(m,n)$ be the impulse response of an IIR filter with corresponding difference equation

$$y(m,n) = 0.01x(m,n) + 0.9(y(m-1,n) + y(m,n-1)) - 0.81y(m-1,n-1)$$

where $x(m,n)$ is the input and $y(m,n)$ is the output

solution

$$y(m,n) - 0.9(y(m-1,n) + y(m,n-1)) + 0.81y(m-1,n-1) = 0.01x(m,n)$$

Do DTFT on the both sides

$$\left[1 - 0.9(e^{-j\mu} + e^{-j\nu}) + 0.81e^{-j(\mu+\nu)}\right] Y\left(e^{j\mu}, e^{j\nu}\right) = 0.01X\left(e^{j\mu}, e^{j\nu}\right)$$

$$H\left(e^{j\mu}, e^{j\nu}\right) \equiv \frac{Y\left(e^{j\mu}, e^{j\nu}\right)}{X\left(e^{j\mu}, e^{j\nu}\right)} = \left(\frac{0.1}{1 - 0.9e^{-j\mu}}\right)\left(\frac{0.1}{1 - 0.9e^{-j\nu}}\right)$$

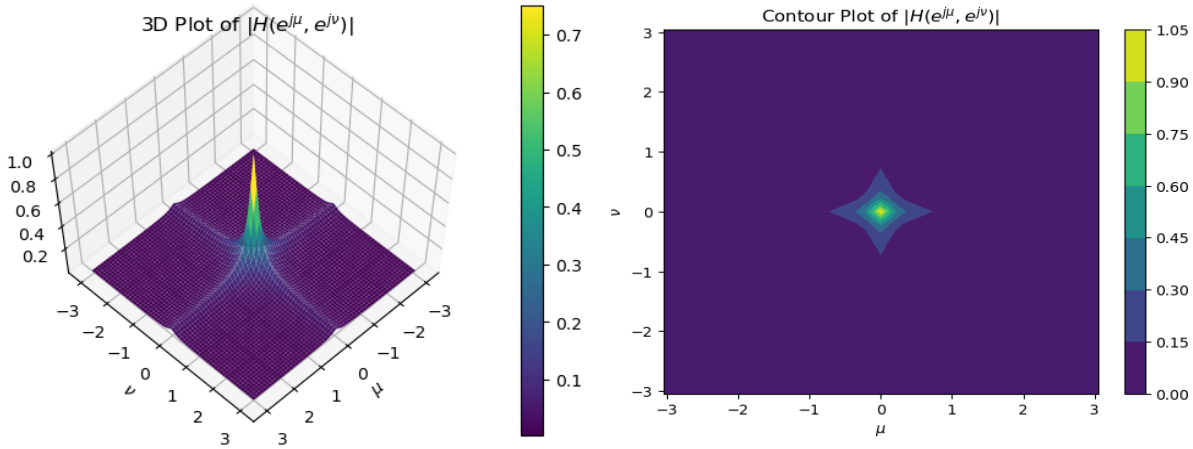
5.2. plot of amplitude for IIR filter $|H\left(e^{j\mu}, e^{j\nu}\right)|$

solution

Run visualization Python script **vis_5.py** with

```
python ./visualize/vis_5.py
```

The left figure below is a 3D mesh plot for $|H\left(e^{j\mu}, e^{j\nu}\right)|$, the right contour figure is a contour plot for $|H\left(e^{j\mu}, e^{j\nu}\right)|$



3D Mesh(left) and Contour(right) Plots for Amplitude of IIR Filter

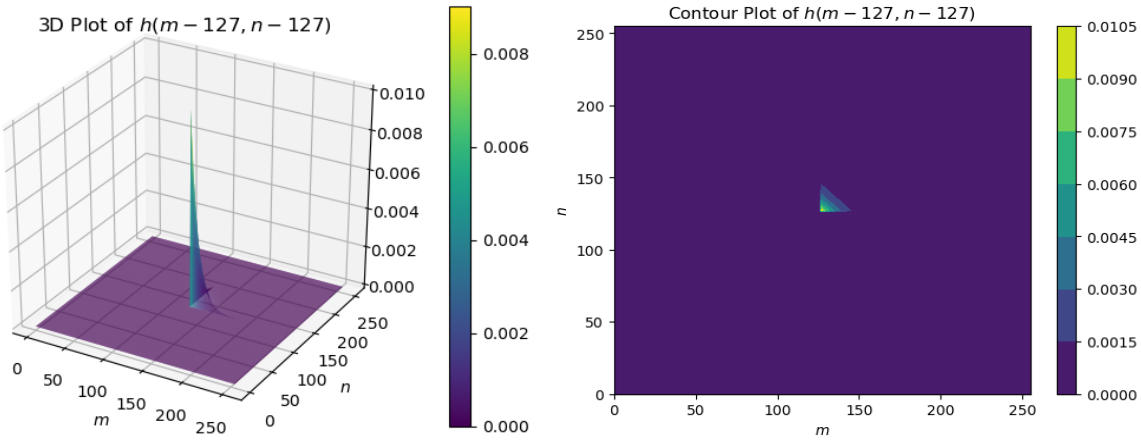
5.3. 256^2 image of the point spread function $h(m-127, n-127)$

solution

Run visualization Python script **vis_5.py** with

```
python ./visualize/vis_5.py
```

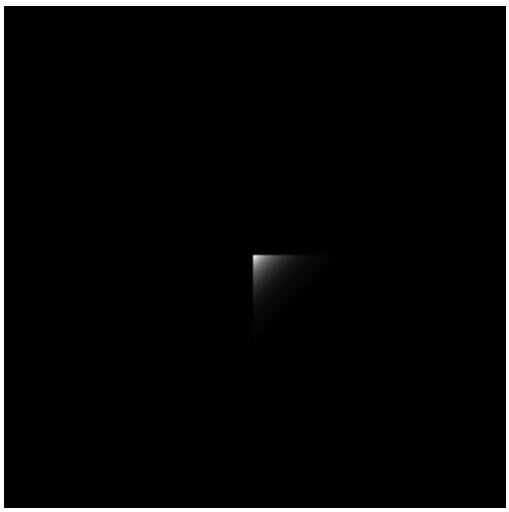
The left figure below is a 3D mesh plot for $h(m-127, n-127)$, the right contour figure is a corresponding contour plot



3D Mesh(left) and Contour(right) Plots for Image of the Point Spread Function

Export the result to a TIFF file using the following scaling

```
im_save=Image.fromarray((255*100*point_spread_2d).astype(np.uint8))
im_save.save("./result/fig_5_3c.tif")
```



Exported TIFF Image for the Point Spread Function

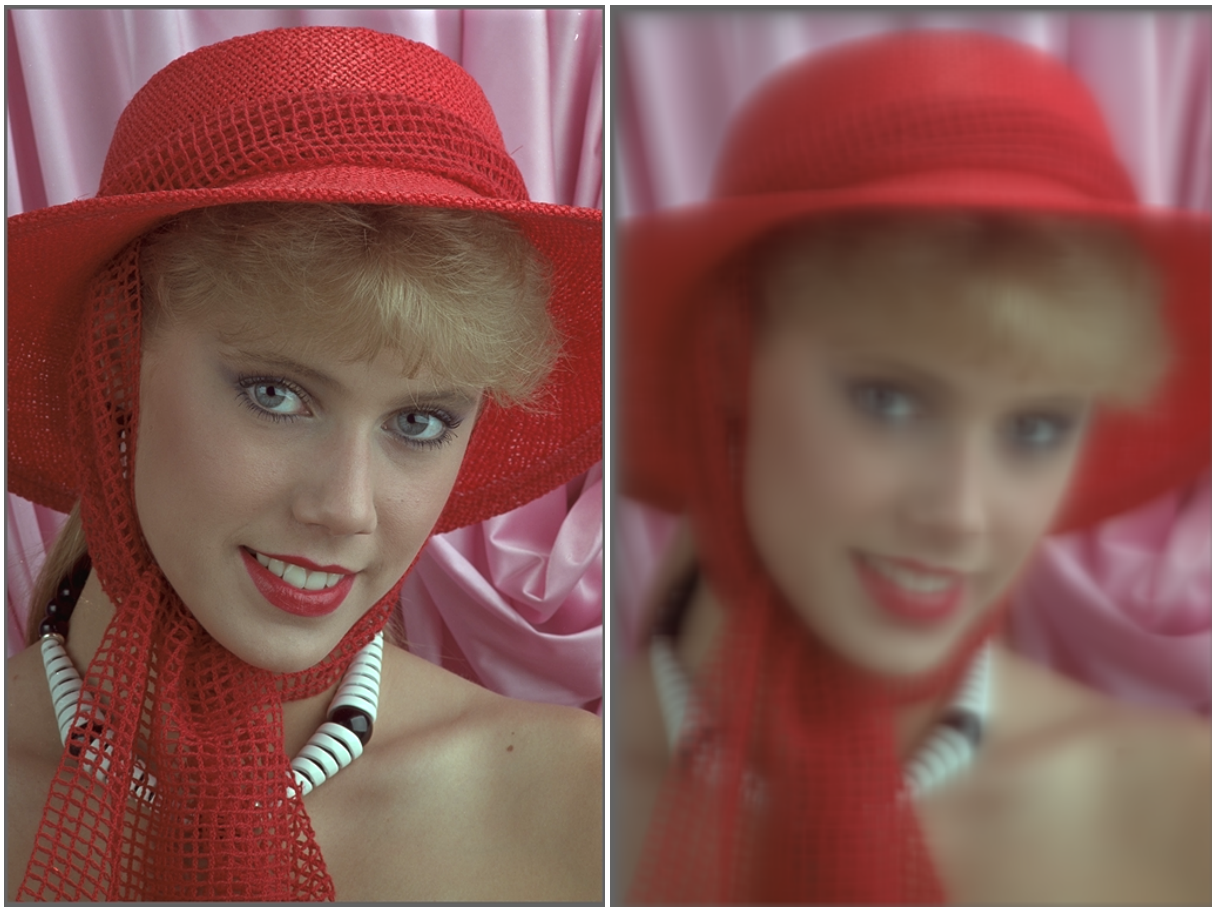
5.4. output filtered color image for input color image `img03.tif`

solution

Run the executable program **soln_5** compiled by **soln_5.c** with

```
./soln_5 img03.tif
```

The input color image `img03.tif` and the corresponding output filtered color image are shown below



Input Color Image **img03.tif***(left) and Output Filtered Color Image(right)*

5.5. listing of C, Python codes

solution

The corresponding function `filter_IIR_lowpass()` for section 5 in **filter.c**

```
void filter_IIR_lowpass(double **a, double**a_t,
                        int16_t W, int16_t H,
                        double pole) {
    double a00 = (1-pole)*(1-pole), b01 = -pole, b11 = pole*pole;
    double **reg_up  = (double **)get_img(W, 1, sizeof(double));
    /* preprocessing: fill the left and up edges */
    for (int16_t j = 0; j < W; j++) { a_t[0][j] = a[0][j];  }
    for (int16_t i = 0; i < H; i++) { a_t[i][0] = a[i][0];  }
    for (int16_t i = 1; i < H; i++) {
        for (int16_t j = 1; j < W; j++) {
            reg_up[0][j] = b01 * a_t[i-1][j] + b11 * a_t[i-1][j-1];
        }
        for (int16_t j = 1; j < W; j++) {
            a_t[i][j] = a00*a[i][j] - b01*a_t[i][j-1] - reg_up[0][j];
        }
    }
    free_img( (void**)reg_up );
}
```

Appendix

C codes for image filtering: `filter.h`, `filter.c`

filter.h

```
#ifndef _FILTER_H_
#define _FILTER_H_

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <math.h>
#include "allocate.h"
#include "typeutil.h"
#include "tiff.h"

void filter_FIR_lowpass(double **a, double**a_t,
                        int16_t W, int16_t H,
                        int16_t kernel_size);
void filter_FIR_sharpen(double **a, double**a_t,
                        int16_t W, int16_t H,
                        int16_t kernel_size,
                        double lambda);
void filter_IIR_lowpass(double **a, double**a_t,
                        int16_t W, int16_t H,
                        double pole);
void assign_img2arr_rgb(struct TIFF_img *img,
                        double **array_red,
                        double **array_green,
                        double **array_blue);
void assign_arr2img_rgb(double **array_red,
                        double **array_green,
                        double **array_blue,
                        struct TIFF_img *img);
#endif /* _FILTER_H_ */
```

filter.c

```
#include "../include/filter.h"


void filter_FIR_lowpass(double **a, double**a_t,
                        int16_t W, int16_t H,
                        int16_t kernel_size) {
    // assert (kernel_size >= 1 &&
    // kernel_size%2 == 1 &&
    // kernel_size <= W &&
    // kernel_size <= H)
    int16_t width = kernel_size / 2;
    double coef_sum = 1.0 / (kernel_size*kernel_size);
    /* allocate memory and set 0 */
    double **a_m = (double **)get_img(W, H, sizeof(double));
    for (int16_t i=0; i < H; i++) {
        memset(a_m[i], 0, W*sizeof(double));
        memset(a_t[i], 0, W*sizeof(double));
    }
    for (int16_t i = 0; i < H; i++) {
        for (int16_t dj = -width; dj <= width; dj++) {
            for (int16_t j = width; j < W-width; j++) {
                a_m[i][j] += a[i][j+dj];
            }
        }
    }
    for (int16_t di = -width; di <= width; di++) {
        for (int16_t i = width; i < H-width; i++) {
            for (int16_t j = width; j < W-width; j++) {
                a_t[i][j] += a_m[i+di][j];
            }
        }
    }
    for (int16_t i = width; i < H-width; i++) {
        for (int16_t j = width; j < W-width; j++) {
            a_t[i][j] *= coef_sum;
        }
    }
    /* fill the border with pixels of input array */
    if (width > 0) {
        for (int16_t i = 0; i < width; i++) {
            for (int16_t j = 0; j < W; j++) {
```

```
                a_t[i][j] = a[i][j];
            }
        }
        for (int16_t i = H-width; i < H; i++) {
            for (int16_t j = 0; j < W; j++) {
                a_t[i][j] = a[i][j];
            }
        }
        for (int16_t i = width; i < H-width; i++) {
            for (int16_t j = 0; j < width; j++) {
                a_t[i][j] = a[i][j];
            }
        }
        for (int16_t i = width; i < H-width; i++) {
            for (int16_t j = W-width; j < W; j++) {
                a_t[i][j] = a[i][j];
            }
        }
    }
    free_img( (void**)a_m );
}


void filter_FIR_sharpen(double **a, double**a_t,
                        int16_t W, int16_t H,
                        int16_t kernel_size,
                        double lambda) {
    filter_FIR_lowpass(a, a_t, W, H, kernel_size);
    int16_t width = kernel_size / 2;
    for (int16_t i = width; i < H-width; i++) {
        for (int16_t j = width; j < W-width; j++) {
            a_t[i][j] *= (-lambda);
            a_t[i][j] += (1+lambda) * a[i][j];
        }
    }
}


void filter_IIR_lowpass(double **a, double**a_t,
                        int16_t W, int16_t H,
                        double pole) {
    double a00 = (1-pole)*(1-pole), b01 = -pole, b11 = pole*pole;
    double **reg_up  = (double **)get_img(W, 1, sizeof(double));
```

```
    /* preprocessing: fill the left and up edges */
    for (int16_t j = 0; j < W; j++) { a_t[0][j] = a[0][j];  }
    for (int16_t i = 0; i < H; i++) { a_t[i][0] = a[i][0];  }
    for (int16_t i = 1; i < H; i++) {
        for (int16_t j = 1; j < W; j++) {
            reg_up[0][j] = b01 * a_t[i-1][j] + b11 * a_t[i-1][j-1];
        }
        for (int16_t j = 1; j < W; j++) {
            a_t[i][j] = a00*a[i][j] - b01*a_t[i][j-1] - reg_up[0][j];
        }
    }
    free_img( (void**)reg_up );
}


void assign_img2arr_rgb(struct TIFF_img *img,
                        double **array_red,
                        double **array_green,
                        double **array_blue) {
    int16_t W, H;
    W = img->width; H = img->height;
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            array_red[i][j] = img->color[0][i][j];
        }
    }
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            array_green[i][j] = img->color[1][i][j];
        }
    }
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            array_blue[i][j] = img->color[2][i][j];
        }
    }
}


void assign_arr2img_rgb(double **array_red,
                        double **array_green,
                        double **array_blue,
                        struct TIFF_img *img) {
```

```
    int16_t W, H;
    W = img->width; H = img->height;
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            int16_t t = round(array_red[i][j]);
            t = (t < 0)? 0: (t > 255)? 255: t;
            img->color[0][i][j] = t;
        }
    }
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            int16_t t = round(array_green[i][j]);
            t = (t < 0)? 0: (t > 255)? 255: t;
            img->color[1][i][j] = t;
        }
    }
    for (int16_t i = 0; i < H; i++ ) {
        for (int16_t j = 0; j < W; j++ ) {
            int16_t t = round(array_blue[i][j]);
            t = (t < 0)? 0: (t > 255)? 255: t;
            img->color[2][i][j] = t;
        }
    }
}
```

C codes for solutions

solution to section 3: `soln_3.c`

```c
/* ECE 637 Image Processing I, Spring 2022
 * @author: Zhankun Luo, luo333@purdue.edu
 * lab 1: Image Filtering
 * solution to section 3
 * run it with: ./soln_3 img03.tif
 **/
#include "../include/tiff.h"
#include "../include/allocate.h"
#include "../include/typeutil.h"
#include "../include/filter.h"
void error(char *name) {
    printf("usage:  %s  image.tif \n\n",name);
    exit(1);
}

int main(int argc, char **argv) {
    int16_t kernel_size = 9;
    if ( argc != 2 ) error( argv[0] );
    FILE *fp;
    struct TIFF_img img, img_out;
    double **arr_r, **arr_g, **arr_b;
    double **arr_r_out, **arr_g_out, **arr_b_out;
    int16_t W, H;
    /* open image file */
    if ( ( fp = fopen( argv[1], "rb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file %s\n", argv[1] );
        exit( 1 );
    }
    /* read image */
    if ( read_TIFF( fp, &img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }
    /* close image file */
    fclose( fp );
    /* check the type of image data: grayscale */
```

```
    if ( img.TIFF_type != 'c' ) {
        fprintf( stderr, "error:  image must be color image\n" );
        exit( 1 );
    }
    /* copy image to array */
    W = img.width; H= img.height;
    get_TIFF( &img_out, H, W, 'c' );
    arr_r = (double **)get_img(W, H, sizeof(double));
    arr_g = (double **)get_img(W, H, sizeof(double));
    arr_b = (double **)get_img(W, H, sizeof(double));
    assign_img2arr_rgb(&img, arr_r, arr_g, arr_b);
    /* filter red, green, blue components of image */
    arr_r_out = (double **)get_img(W, H, sizeof(double));
    arr_g_out = (double **)get_img(W, H, sizeof(double));
    arr_b_out = (double **)get_img(W, H, sizeof(double));
    /* clip to [0, 255] then assign values of arrays to image */
    filter_FIR_lowpass(arr_r, arr_r_out, W, H, kernel_size);
    filter_FIR_lowpass(arr_g, arr_g_out, W, H, kernel_size);
    filter_FIR_lowpass(arr_b, arr_b_out, W, H, kernel_size);
    assign_arr2img_rgb(arr_r_out, arr_g_out, arr_b_out, &img_out);
    /* open output image file */
    if ( ( fp = fopen ( "../result/fig_3_4.tif", "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file fig_3_4.tif\n");
        exit( 1 );
    }
    /* write output image */
    if ( write_TIFF( fp, &img_out ) ) {
        fprintf( stderr, "error writing TIFF file fig_3_4.tif\n");
        exit( 1 );
    }
    /* close output image file */
    fclose( fp );
    /* de-allocate memory */
    free_TIFF( &(img) );
    free_TIFF( &(img_out) );
    free_img( (void**)arr_r );
    free_img( (void**)arr_g );
    free_img( (void**)arr_b );
    free_img( (void**)arr_r_out );
    free_img( (void**)arr_g_out );
    free_img( (void**)arr_b_out );
    return(0);
}
```

solution to section 4: `soln_4.c`

```c
/* ECE 637 Image Processing I, Spring 2022
 * @author: Zhankun Luo, luo333@purdue.edu
 * lab 1: Image Filtering
 * solution to section 4
 * run it with: ./soln_4 1.5 imgblur.tif
 **/

#include "../include/tiff.h"
#include "../include/allocate.h"
#include "../include/typeutil.h"
#include "../include/filter.h"
void error(char *name) {
    printf("usage:  %s  image.tif \n\n",name);
    exit(1);
}

int main(int argc, char **argv) {
    int16_t kernel_size = 5;
    double lambda;
    if ( argc != 3 ) error( argv[0] );
    FILE *fp;
    struct TIFF_img img, img_out;
    double **arr_r, **arr_g, **arr_b;
    double **arr_r_out, **arr_g_out, **arr_b_out;
    int16_t W, H;
    /* set lambda for sharpening filter, open image file */
    sscanf(argv[1], "%lf", &lambda);
    if ( ( fp = fopen( argv[2], "rb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file %s\n", argv[1] );
        exit( 1 );
    }
    /* read image */
    if ( read_TIFF( fp, &img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }
    /* close image file */
    fclose( fp );
    /* check the type of image data: grayscale */
```

```
    if ( img.TIFF_type != 'c' ) {
        fprintf( stderr, "error:  image must be color image\n" );
        exit( 1 );
    }
    /* copy image to array */
    W = img.width; H= img.height;
    get_TIFF( &img_out, H, W, 'c' );
    arr_r = (double **)get_img(W, H, sizeof(double));
    arr_g = (double **)get_img(W, H, sizeof(double));
    arr_b = (double **)get_img(W, H, sizeof(double));
    assign_img2arr_rgb(&img, arr_r, arr_g, arr_b);
    /* filter red, green, blue components of image */
    arr_r_out = (double **)get_img(W, H, sizeof(double));
    arr_g_out = (double **)get_img(W, H, sizeof(double));
    arr_b_out = (double **)get_img(W, H, sizeof(double));
    /* clip to [0, 255] then assign values of arrays to image */
    filter_FIR_sharpen(arr_r, arr_r_out, W, H, kernel_size, lambda);
    filter_FIR_sharpen(arr_g, arr_g_out, W, H, kernel_size, lambda);
    filter_FIR_sharpen(arr_b, arr_b_out, W, H, kernel_size, lambda);
    assign_arr2img_rgb(arr_r_out, arr_g_out, arr_b_out, &img_out);
    /* open output image file */
    if ( ( fp = fopen ( "../result/fig_4_6.tif", "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file fig_4_6.tif\n");
        exit( 1 );
    }
    /* write output image */
    if ( write_TIFF( fp, &img_out ) ) {
        fprintf( stderr, "error writing TIFF file fig_4_6.tif\n");
        exit( 1 );
    }
    /* close output image file */
    fclose( fp );
    /* de-allocate memory */
    free_TIFF( &(img) );
    free_TIFF( &(img_out) );
    free_img( (void**)arr_r );
    free_img( (void**)arr_g );
    free_img( (void**)arr_b );
    free_img( (void**)arr_r_out );
    free_img( (void**)arr_g_out );
    free_img( (void**)arr_b_out );
    return(0);
}
```

solution to section 5: `soln_5.c`

```c
/* ECE 637 Image Processing I, Spring 2022
 * @author: Zhankun Luo, luo333@purdue.edu
 * lab 1: Image Filtering
 * solution to section 5
 * run it with: ./soln_5 img03.tif
 **/

#include "../include/tiff.h"
#include "../include/allocate.h"
#include "../include/typeutil.h"
#include "../include/filter.h"
void error(char *name) {
    printf("usage:  %s  image.tif \n\n",name);
    exit(1);
}

int main(int argc, char **argv) {
    double pole = 0.9;
    if ( argc != 2 ) error( argv[0] );
    FILE *fp;
    struct TIFF_img img, img_out;
    double **arr_r, **arr_g, **arr_b;
    double **arr_r_out, **arr_g_out, **arr_b_out;
    int16_t W, H;
    /* open image file */
    if ( ( fp = fopen( argv[1], "rb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file %s\n", argv[1] );
        exit( 1 );
    }
    /* read image */
    if ( read_TIFF( fp, &img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }
    /* close image file */
    fclose( fp );
    /* check the type of image data: grayscale */
    if ( img.TIFF_type != 'c' ) {
        fprintf( stderr, "error:  image must be color image\n" );
```

```
        exit( 1 );
    }
    /* copy image to array */
    W = img.width; H= img.height;
    get_TIFF( &img_out, H, W, 'c' );
    arr_r = (double **)get_img(W, H, sizeof(double));
    arr_g = (double **)get_img(W, H, sizeof(double));
    arr_b = (double **)get_img(W, H, sizeof(double));
    assign_img2arr_rgb(&img, arr_r, arr_g, arr_b);
    /* filter red, green, blue components of image */
    arr_r_out = (double **)get_img(W, H, sizeof(double));
    arr_g_out = (double **)get_img(W, H, sizeof(double));
    arr_b_out = (double **)get_img(W, H, sizeof(double));
    /* clip to [0, 255] then assign values of arrays to image */
    filter_IIR_lowpass(arr_r, arr_r_out, W, H, pole);
    filter_IIR_lowpass(arr_g, arr_g_out, W, H, pole);
    filter_IIR_lowpass(arr_b, arr_b_out, W, H, pole);
    assign_arr2img_rgb(arr_r_out, arr_g_out, arr_b_out, &img_out);
    /* open output image file */
    if ( ( fp = fopen ( "../result/fig_5_4.tif", "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file fig_5_4.tif\n");
        exit( 1 );
    }
    /* write output image */
    if ( write_TIFF( fp, &img_out ) ) {
        fprintf( stderr, "error writing TIFF file fig_5_4.tif\n");
        exit( 1 );
    }
    /* close output image file */
    fclose( fp );
    /* de-allocate memory */
    free_TIFF( &(img) );
    free_TIFF( &(img_out) );
    free_img( (void**)arr_r );
    free_img( (void**)arr_g );
    free_img( (void**)arr_b );
    free_img( (void**)arr_r_out );
    free_img( (void**)arr_g_out );
    free_img( (void**)arr_b_out );
    return(0);
}
```

Python codes for visualization functions: `utils.py`

```
from math import pi, sin, cos, sqrt, log
from typing import Tuple, Union
import matplotlib.pyplot as plt
import numpy as np
import sys
max_exp, min_exp = sys.float_info.max_exp, sys.float_info.min_exp

def memoize(f):
   cache = {}
   def memoizedFunction(*args):
      if args not in cache:
         cache[args] = f(*args)
      return cache[args]
   memoizedFunction.cache = cache
   return memoizedFunction

sin = memoize(sin)
cos = memoize(cos)

def rect_1d(N: int, A: float = None) -> dict:
    assert N%2 == 1
    A = 1./ N if A is None else A # amplitude of rect signal
    width = (N >> 1)
    return dict(zip(list(range(-width, width+1)), [A] * N))

def calc_spectrum_1d(coef: Union[dict, Tuple[dict, dict]],
                     list_omega: list) -> Tuple[list, list]:
    if isinstance(coef, dict):
        return calc_spectrum_1d_dict(coef, list_omega)
    else:
        return calc_spectrum_1d_tuple(coef, list_omega)

def calc_spectrum_1d_dict(dict_coef: dict,
                          list_omega: list) -> Tuple[list, list]:
    func_re = lambda omega: \
        sum( [ coef * cos( omega * n )
               for n, coef in dict_coef.items() ] )
    func_im = lambda omega: \
        - sum( [ coef * sin( omega * n )
```

```
                for n, coef in dict_coef.items() ] )
    list_re = list( map(func_re, list_omega) )
    list_im = list( map(func_im, list_omega) )
    return list_re, list_im

def calc_spectrum_1d_tuple(tuple_coef: Tuple[dict, dict],
                           list_omega) -> Tuple[list, list]:
    list_re_num, list_im_num = \
        calc_spectrum_1d_dict(tuple_coef[0], list_omega)
    list_re_den, list_im_den = \
        calc_spectrum_1d_dict(tuple_coef[1], list_omega)
    list_re = [(a1*a2 + b1*b2) / (a2*a2 + b2*b2)
               for a1, b1, a2, b2 in
               list(zip(list_re_num, list_im_num,
                        list_re_den, list_im_den))]
    list_im = [(-a1*b2 + a2*b1) / (a2*a2 + b2*b2)
               for a1, b1, a2, b2 in
               list(zip(list_re_num, list_im_num,
                        list_re_den, list_im_den))]
    return list_re, list_im

def calc_omega_1d(num_point: int = 32) -> list:
    return [pi * i / num_point for i in range(num_point)]

def calc_mag_1d(list_re: list, list_im: list) -> list:
    return [sqrt(re*re+im*im) for re, im
            in list(zip(list_re, list_im))]

def get_mag_1d(coef: Union[dict, Tuple[dict, dict]],
               num_point: int = 32) -> Tuple[list, list]:
    list_omega = calc_omega_1d(num_point)
    list_re, list_im = calc_spectrum_1d(coef, list_omega)
    list_mag = calc_mag_1d(list_re, list_im)
    list_omega_neg = [-omega for omega in list_omega[-1:0:-1]]
    return list(reversed(list_mag))[:-1] + list_mag, \
        list_omega_neg + list_omega

def expand_1d_2d(a: list, b: list) -> np.ndarray:
    a = np.array(a).reshape((-1, 1))
    b = np.array(b).reshape((1, -1))
    return a @ b

def plot_3d(list_mag: Union[list, np.ndarray],
```

```
            list_omega: list, symbol='H') -> None:
    X, Y = np.meshgrid(list_omega, list_omega)
    Z = expand_1d_2d(list_mag, list_mag) \
        if isinstance(list_mag, list) else list_mag
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.view_init(60, 45)
    cp = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
    fig.colorbar(cp)
    str_title = r'3D Plot of $|' + symbol \
        + r'(e^{j \mu}, e^{j \nu})|$'
    ax.set_title(str_title)
    ax.set_xlabel(r'$\mu$')
    ax.set_ylabel(r'$\nu$')

def plot_contour(list_mag: Union[list, np.ndarray],
                 list_omega, symbol='H') -> None:
    X, Y = np.meshgrid(list_omega, list_omega)
    Z = expand_1d_2d(list_mag, list_mag) \
        if isinstance(list_mag, list) else list_mag
    fig, ax=plt.subplots(1, 1)
    cp = ax.contourf(X, Y, Z)
    fig.colorbar(cp)
    str_title = r'Contour Plot of $|' + symbol \
        + r'(e^{j \mu}, e^{j \nu})|$'
    ax.set_title(str_title)
    ax.set_xlabel(r'$\mu$')
    ax.set_ylabel(r'$\nu$')

def _add(a: dict, b: dict) -> dict:
    d = dict(a)
    for k, v in b.items():
        d[k] = d[k] + v if k in d else v
    return d

def _multiply(coef: float, d: dict) -> dict:
    d_tmp = {}
    for k, v in d.items():
        d_tmp[k] = coef * v
    return d_tmp

def log_point_spread_func(pole: float, length: int) -> list():
```

```
    delay = (length-1) >> 1
    assert (delay >= 0 and length > delay)
    log_response = [log(1-pole) + log(pole) * n for n
                    in range(length-delay)]
    return [None] * delay + log_response

def expand_point_spread_1d_2d(log_response: list) -> np.ndarray:
    length= len(log_response)
    delay = next(i for i, v in enumerate(log_response)
                 if v is not None)
    point_spread_2d = np.zeros((length, length))
    l = np.array(log_response[delay:]).reshape((-1, 1))
    l_2d = l + l.T
    l_2d[l_2d > max_exp] = max_exp
    l_2d[l_2d < min_exp] = min_exp
    point_spread_2d[delay:, delay:] = np.exp(l_2d)
    return point_spread_2d

def plot_point_spread_3d(point_spread_2d: np.ndarray,
                         symbol='h') -> None:
    M, N = point_spread_2d.shape
    delay_m, delay_n = (M-1)>>1, (N-1)>>1
    X, Y = np.meshgrid(list(range(M)), list(range(N)))
    Z = point_spread_2d
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    # ax.view_init(60, 45)
    cp = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
    fig.colorbar(cp)
    str_title = r'3D Plot of $' + symbol \
        + r'(m-%s, n-%s)$'%(delay_m , delay_n)
    ax.set_title(str_title)
    ax.set_xlabel(r'$m$')
    ax.set_ylabel(r'$n$')

def plot_point_spread_contour(point_spread_2d: np.ndarray,
                              symbol='h') -> None:
    M, N = point_spread_2d.shape
    delay_m, delay_n = (M-1)>>1, (N-1)>>1
    X, Y = np.meshgrid(list(range(M)), list(range(N)))
    Z = point_spread_2d
    fig, ax=plt.subplots(1, 1)
```

```
    cp = ax.contourf(X, Y, Z)
    fig.colorbar(cp)
    str_title = r'Contour Plot of $' + symbol \
        + r'(m-%s, n-%s)$'%(delay_m , delay_n)
    ax.set_title(str_title)
    ax.set_xlabel(r'$m$')
    ax.set_ylabel(r'$n$')
```

Python codes for visualizations

visualization to section 3: `vis_3.py`

```
from utils import rect_1d, get_mag_1d, plot_3d, plot_contour, plt

if __name__ == "__main__":
    h = rect_1d(9)
    list_mag, list_omega = get_mag_1d(h)
    plot_3d(list_mag, list_omega)
    plt.savefig("./result/fig_3_2a.png", bbox_inches='tight')
    plt.show()
    plot_contour(list_mag, list_omega)
    plt.savefig("./result/fig_3_2b.png", bbox_inches='tight')
    plt.show()
```

visualization to section 4: `vis_4.py`

```
from utils import rect_1d, get_mag_1d, plot_3d, plot_contour, plt
from utils import calc_spectrum_1d, expand_1d_2d
import numpy as np
if __name__ == "__main__":
    h, lamda = rect_1d(5), 1.5
    list_mag, list_omega = get_mag_1d(h)
    plot_3d(list_mag, list_omega)
    plt.savefig("./result/fig_4_3a.png", bbox_inches='tight')
    plt.show()
    plot_contour(list_mag, list_omega)
    plt.savefig("./result/fig_4_3b.png", bbox_inches='tight')
    plt.show()
    H_re, H_im = calc_spectrum_1d(h, list_omega)
    H_1d = [a + 1j*b for a, b in list(zip(H_re, H_im))]
    H_2d = expand_1d_2d(H_1d, H_1d)
    G_2d = (1+lamda) - lamda * H_2d
    list_mag = np.absolute(G_2d)
    plot_3d(list_mag, list_omega, 'G')
    plt.savefig("./result/fig_4_4a.png", bbox_inches='tight')
    plt.show()
    plot_contour(list_mag, list_omega, 'G')
    plt.title(r'for $\lambda=%s$'%(lamda), loc="right")
    plt.savefig("./result/fig_4_4b.png", bbox_inches='tight')
    plt.show()
```

visualization to section 5: `vis_5.py`

```
from utils import rect_1d, get_mag_1d, plot_3d, plot_contour, plt
from utils import log_point_spread_func, expand_point_spread_1d_2d,\
    plot_point_spread_3d, plot_point_spread_contour
from PIL import Image
import numpy as np
if __name__ == "__main__":
    pole = 0.9
    h = ({0: 1-pole}, {0: 1, 1: -pole}) # H(z) = 0.1 / (1 - 0.9
z^{-1})
    list_mag, list_omega = get_mag_1d(h)
    plot_3d(list_mag, list_omega)
    plt.savefig("./result/fig_5_2a.png", bbox_inches='tight')
    plt.show()
    plot_contour(list_mag, list_omega)
    plt.savefig("./result/fig_5_2b.png", bbox_inches='tight')
    plt.show()
    length = 256
    log_response = log_point_spread_func(pole, length)
    point_spread_2d = expand_point_spread_1d_2d(log_response)
    plot_point_spread_3d(point_spread_2d)
    plt.savefig("./result/fig_5_3a.png", bbox_inches='tight')
    plt.show()
    plot_point_spread_contour(point_spread_2d)
    plt.savefig("./result/fig_5_3b.png", bbox_inches='tight')
    plt.show()
    im_save = \
        Image.fromarray((255*100*point_spread_2d).astype(np.uint8))
    im_save.save("./result/fig_5_3c.tif")
```