

# ECE637 : Digital Image Processing : Lab1

Sriranga R

January 19, 2023

## 1 C Programming

No submission due for this.

## 2 Display and Export Images in Python

No submission due for this.

### 3 FIR Low Pass Filter

#### 3.1 Derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$

Problem 3.

$$h(m, n) = \begin{cases} 1/21, & |m| \leq 4, |n| \leq 4 \\ 0, & \text{otherwise.} \end{cases}$$

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m, n) e^{-j(\mu m + \nu n)}.$$

$$= \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{21} e^{-j\mu m} e^{-j\nu n}.$$

$$= \frac{1}{21} \sum_{n=-4}^{4} e^{-j\nu n} \sum_{m=-4}^{4} e^{-j\mu m}.$$

$$\text{Consider a GP. } \sum_{n=0}^N r^n = \frac{a(1-r^N)}{(1-r)}.$$

$$r = e^{-j\nu}, \quad a = e^{4j\nu}.$$

$$\therefore \sum_{n=-4}^{4} e^{-j\nu n} = \frac{e^{4j\nu}(1 - e^{-j4\nu})}{1 - e^{-j\nu}}.$$

$$= \frac{e^{4j\nu} - e^{-5j\nu}}{1 - e^{-j\nu}}.$$

$$= \frac{e^{4j\nu}(1 - e^{-9j\nu})}{1 - e^{-j\nu}}$$

$$= \frac{e^{4j\nu} e^{-9/2j\nu} [e^{9/2j\nu} - e^{-9/2j\nu}]}{1 - e^{-j\nu} [e^{j\nu} - e^{-j\nu}]}.$$

$$= \frac{e^{-j\nu} (2j \sin \frac{9}{2}\nu)}{2j \sin \frac{1}{2}\nu}$$

$$= \frac{\sin \frac{9}{2}\nu}{\sin \frac{1}{2}\nu}.$$

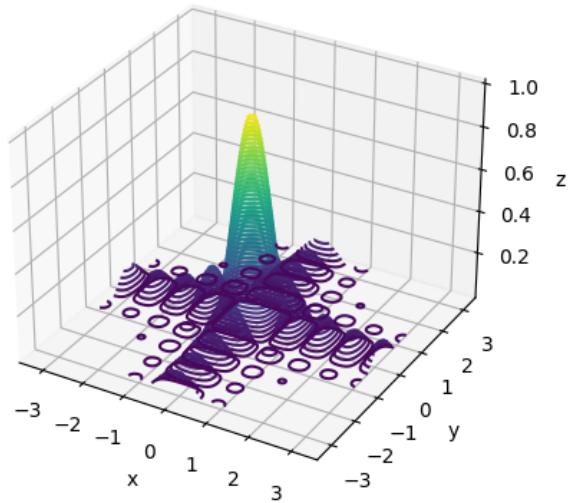
$$\therefore \frac{1}{21} \sum_{n=-4}^{4} e^{-j\nu n} \sum_{m=-4}^{4} e^{-j\mu m}$$

$$= \frac{1}{21} \frac{\sin \frac{9}{2}\nu}{\sin \frac{1}{2}\nu} \cdot \frac{\sin \frac{9}{2}\mu}{\sin \frac{1}{2}\mu}.$$

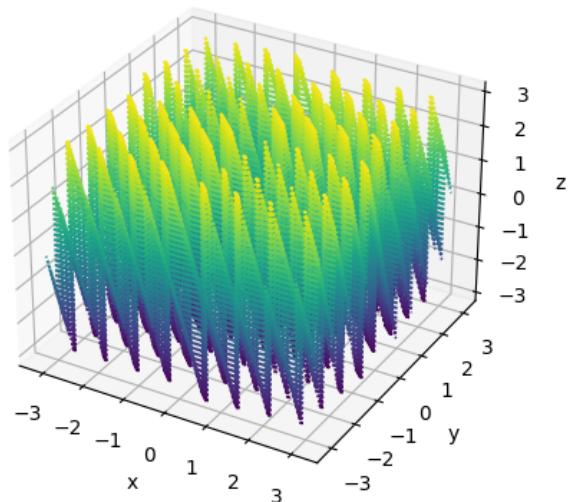
=====.

### 3.2 Plot of $|H(e^{j\mu}, e^{j\nu})|$

Magnitude of H



Phase of H



### 3.3 Original Image - img03.tif



### 3.4 The filtered color image.



### 3.5 C Code

```
1
2 #include <math.h>
3 #include "tiff.h"
4 #include "allocate.h"
```

```

5 #include "randlib.h"
6 #include "typeutil.h"
7 #include "defs.h"
8
9 void error(char* );
10 int8_t filterVal(struct TIFF_img* img, int row, int col);
11
12 int main(int argc, char** argv)
13 {
14     FILE *fp;
15     struct TIFF_img input_img, color_img;
16
17     if ( argc != 2 ) error( argv[0] );
18
19     /* open image file */
20     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
21         fprintf ( stderr, "cannot open file %s\n", argv[1] );
22         exit ( 1 );
23     }
24
25     /* read image */
26     if ( read_TIFF ( fp, &input_img ) ) {
27         fprintf ( stderr, "error reading file %s\n", argv[1] );
28         exit ( 1 );
29     }
30
31     /* close image file */
32     fclose ( fp );
33     //int32_t output_height = input_img.height + (2 * y_pad);
34     //int32_t output_width = input_img.width + (2 * x_pad);
35     get_TIFF ( &color_img, input_img.height, input_img.width, 'c' )
36     ;
37     printf("Input Height = %d, Input width = %d\n", input_img.
38     height, input_img.width);
39
40     double** filter = (double**)calloc(9,sizeof(double*));
41     for(int i = 0; i < 9; i++)
42     {
43         filter[i] = (double*)calloc(9, sizeof(double));
44     }
45
46     for(int i = 0; i < 9; i++)
47     {
48         for(int j = 0; j < 9; j++)
49         {
50             filter[i][j] = 1.0/81;
51         }
52     }
53     convolution2D(&input_img, &color_img, 9, 9, filter);
54
55     if ( ( fp = fopen ( "color_output.tif", "wb" ) ) == NULL ) {
56         fprintf ( stderr, "cannot open file color.tif\n");
57         exit ( 1 );
58     }
59
/* write color image */
if ( write_TIFF ( fp, &color_img ) ) {

```

```

60     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
61     exit ( 1 );
62 }
63
64 fclose ( fp );
65
66 free_TIFF ( &(input_img) );
67 free_TIFF ( &(color_img) );
68
69 return 0;
70 }
71
72 void error(char *name)
73 {
74     printf("usage: %s image.tiff \n\n",name);
75     printf("this program reads in a 24-bit color TIFF image.\n");
76     printf("It then horizontally filters the green component, adds
77 noise,\n");
78     printf("and writes out the result as an 8-bit image\n");
79     printf("with the name 'green.tiff'.\n");
80     printf("It also generates an 8-bit color image,\n");
81     printf("that swaps red and green components from the input
82 image");
83     exit(1);
84 }
```

defs.c consists the implementation of convolution2D.

```

1 #include "defs.h"
2
3 // Convolution in 2D
4 // y[m,n] =
5 // Sum(u)
6 //
7 //      Sum(v)
8 //      {
9 //          h(u,v) * x(m-u, n-v)
10 //      }
11 // }
12
13 void convolution2D(struct TIFF_img* inputImg, struct TIFF_img*
14 outputImg, int filterHeight, int filterWidth,
15 double** filter)
16 {
17     int fhIter = (filterHeight - 1)/2;
18     int fwIter = (filterWidth - 1)/2;
19     int imageHeight = inputImg->height;
20     int imageWidth = inputImg->width;
21
22     double red, green, blue;
23
24     for(int i = 0; i < imageHeight; i++)
25     {
26         for(int j = 0; j < imageWidth; j++)
27         {
28             red = 0;
29             green = 0;
30             blue = 0;
```

```

30         for(int row = -fhIter; row <= fhIter; row++)
31     {
32         for(int col = -fwIter; col <= fwIter; col++)
33     {
34             int index_x = i - row;
35             int index_y = j - col;
36             if(index_x < imageHeight && index_y <
imageWidth && index_x >= 0 && index_y >= 0)
37             {
38                 red += filter[row + fhIter][col + fwIter] *
inputImg->color[0][index_x][index_y];
39                 green += filter[row + fhIter][col + fwIter]
* inputImg->color[1][index_x][index_y];
40                 blue += filter[row + fhIter][col + fwIter]
* inputImg->color[2][index_x][index_y];
41             }
42         }
43         outputImg->color[0][i][j] = limitPixel(red);
44         outputImg->color[1][i][j] = limitPixel(green);
45         outputImg->color[2][i][j] = limitPixel(blue);
46     }
47 }
48 }
49 }
50
51 uint8_t limitPixel(double pixel)
52 {
53     uint8_t colorVal;
54     if(pixel > 255)
55     {
56         colorVal = 255;
57     }
58     else if(pixel < 0)
59     {
60         colorVal = 0;
61     }
62     else
63     {
64         colorVal = (uint8_t)pixel;
65     }
66     return colorVal;
67 }
```

## 4 FIR Sharpening Filter

4.1 Derivation of the analytical expression for  $H(e^{j\mu}, e^{j\nu})$

4.2 Derivation of the analytical expression for  $G(e^{j\mu}, e^{j\nu})$

Problem 4.

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

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-2}^2 \sum_{m=-2}^2 h(m, n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-2}^2 \sum_{m=-2}^2 \frac{1}{25} e^{-j(\mu m + \nu n)}$$

$$= \frac{1}{25} \sum_{n=-2}^2 e^{-j\nu n} \sum_{m=-2}^2 e^{-j\mu m}$$

$$\sum_{n=-2}^2 e^{-j\nu n} = \frac{e^{2j\nu} (1 - e^{-5j\nu})}{1 - e^{-j\nu}}$$

$$= e^{2j\nu} e^{-5j_2\nu} (e^{j_2\nu} - e^{-j_2\nu})$$

$$= \frac{e^{-j_2\nu}}{e^{j_2\nu}} \frac{2j \sin(j_2\nu)}{\sin(j_2\nu)}$$

$$\therefore H(e^{j\mu}, e^{j\nu}) = \frac{1}{25} \frac{\sin(j_2\mu)}{\sin(j_2\nu)} \frac{\sin(j_2\nu)}{\sin(j_2\mu)}$$

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

$$\therefore G(e^{j\mu}, e^{j\nu}) = 1 + 1.5(1 - H(e^{j\mu}, e^{j\nu}))$$

$$= 1 + 1.5 \left( 1 - \frac{1}{25} \frac{\sin(j_2\mu)}{\sin(j_2\nu)} \frac{\sin(j_2\nu)}{\sin(j_2\mu)} \right)$$

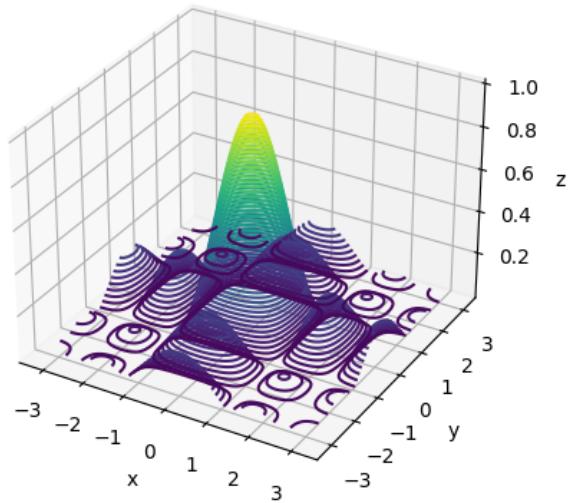
$$= 1 + 1.5 \left( 1 - \frac{1}{25} \frac{\sin(j_2\mu)}{\sin(j_2\nu)} \frac{\sin(j_2\nu)}{\sin(j_2\mu)} \right)$$

$$\underline{(m\mu + n\nu)}, \underline{S \text{ (constant)}}, \underline{= (v_0, v_0) H}$$

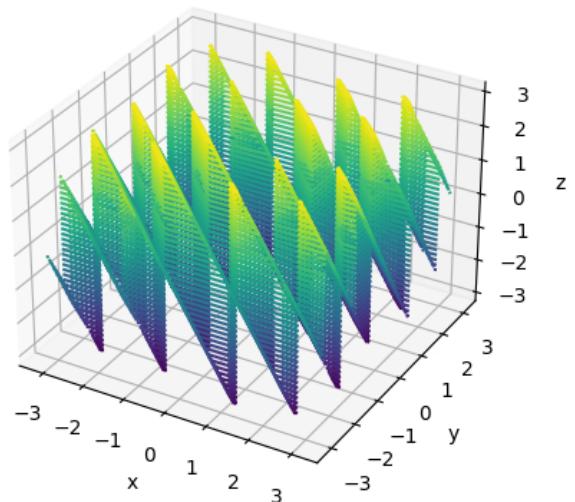
$$(m\mu + n\nu)$$

### 4.3 Plot of $|H(e^{j\mu}, e^{j\nu})|$

Magnitude of H

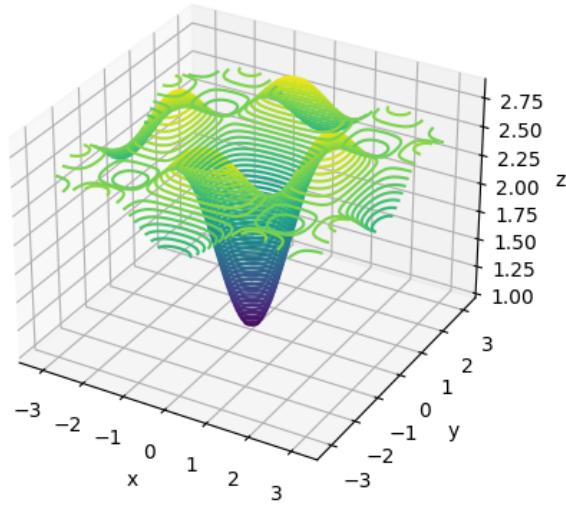


Phase of H

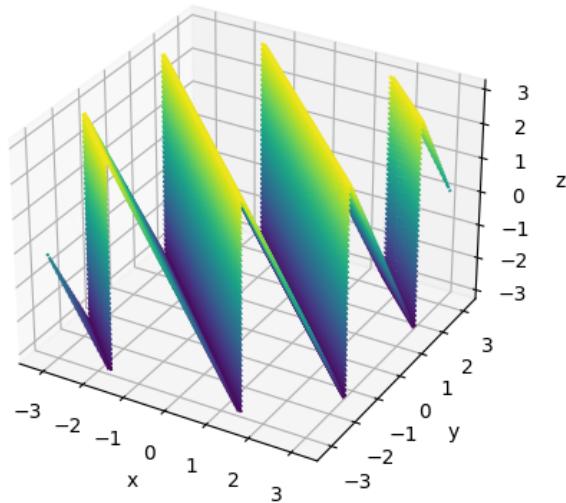


#### 4.4 Plot of $|G(e^{j\mu}, e^{j\nu})|$

Magnitude of G



Phase of G



4.5 The input color image imgblur.tif.



4.6 The output sharpened color image for  $\lambda = 1.5$



4.7 C code.

```
1
2 #include <math.h>
3 #include "tiff.h"
4 #include "allocate.h"
```

```

5 #include "randlib.h"
6 #include "typeutil.h"
7 #include "defs.h"
8
9 void error(char* );
10 int8_t filterVal(struct TIFF_img* img, int row, int col);
11
12 int main(int argc, char** argv)
13 {
14     FILE *fp;
15     struct TIFF_img input_img, color_img;
16
17     if ( argc != 3 ) error( argv[0] );
18     float lambda = atof(argv[2]);
19
20     /* open image file */
21     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
22         fprintf ( stderr, "cannot open file %s\n", argv[1] );
23         exit ( 1 );
24     }
25
26     /* read image */
27     if ( read_TIFF ( fp, &input_img ) ) {
28         fprintf ( stderr, "error reading file %s\n", argv[1] );
29         exit ( 1 );
30     }
31
32     /* close image file */
33     fclose ( fp );
34     //int32_t output_height = input_img.height + (2 * y_pad);
35     //int32_t output_width = input_img.width + (2 * x_pad);
36     get_TIFF ( &color_img, input_img.height, input_img.width, 'c' )
37 ;
38     double** filter = (double**)calloc(5,sizeof(double*));
39     for(int i = 0; i < 5; i++)
40     {
41         filter[i] = (double*)calloc(5, sizeof(double));
42     }
43     double delta = 0;
44     for(int i = 0; i < 5; i++)
45     {
46         for(int j = 0; j < 5; j++)
47         {
48             if(i == 2 && j == 2)
49             {
50                 delta = 1.0;
51             }
52             else
53             {
54                 delta = 0.0;
55             }
56             filter[i][j] = delta + lambda * (delta - (1.0/25));
57         }
58     }
59     convolution2D(&input_img, &color_img, 5, 5, filter);
60     if ( ( fp = fopen ( "color_sharp.tif", "wb" ) ) == NULL ) {

```

```

61     fprintf ( stderr, "cannot open file color.tif\n");
62     exit ( 1 );
63 }
64
65 /* write color image */
66 if ( write_TIFF ( fp, &color_img ) ) {
67     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
68     exit ( 1 );
69 }
70
71 fclose ( fp );
72
73 free_TIFF ( &(input_img) );
74 free_TIFF ( &(color_img) );
75
76 return 0;
77 }
78
79 void error(char *name)
80 {
81     printf("usage: %s image.tiff \n\n",name);
82     printf("this program reads in a 24-bit color TIFF image.\n");
83     printf("It then horizontally filters the green component, adds
noise,\n");
84     printf("and writes out the result as an 8-bit image\n");
85     printf("with the name 'green.tiff'.\n");
86     printf("It also generates an 8-bit color image,\n");
87     printf("that swaps red and green components from the input
image");
88     exit(1);
89 }
```

## 5 IIR Filter

### 5.1 Derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$

Problem 5

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

Take the Z-transform.

$$Y(z_1, z_2) = 0.01X(z_1, z_2) + 0.9z_1^{-1}Y(z_1, z_2) + 0.9z_2^{-1}Y(z_1, z_2) \\ - 0.81z_1^{-1}z_2^{-1}Y(z_1, z_2).$$

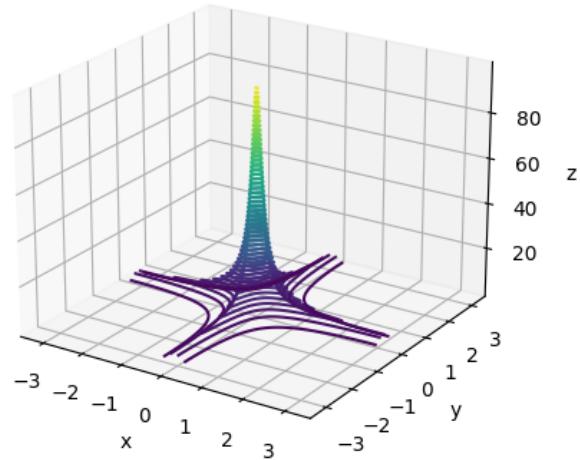
$$\therefore H(z_1, z_2) = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}}$$

$$z_1 = e^{-j\mu} \quad z_2 = e^{-j\nu}$$

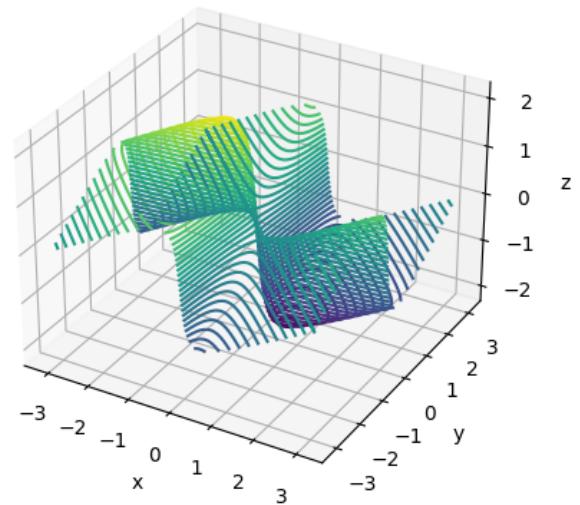
$$\therefore H(e^{j\mu}, e^{j\nu}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j(\mu+\nu)}} \\ \underline{\hspace{10em}}.$$

## 5.2 Plot of $|H(e^{j\mu}, e^{j\nu})|$

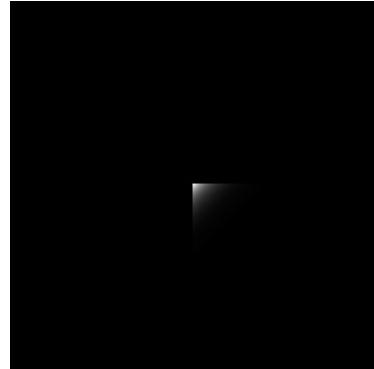
Magnitude of H



Phase of H



### 5.3 An image of the point spread function



### 5.4 The filtered output color image.



### 5.5 C code

```
1 #include <math.h>
2 #include "tiff.h"
3 #include "allocate.h"
4 #include "randlib.h"
5 #include "typeutil.h"
6 #include "defs.h"
7
8 void error(char*);
9 int8_t filterVal(struct TIFF_img* img, int row, int col);
10
11
```

```

12 int main(int argc, char** argv)
13 {
14     FILE *fp;
15     struct TIFF_img input_img, color_img;
16
17     if ( argc != 2 ) error( argv[0] );
18
19     /* open image file */
20     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
21         fprintf ( stderr, "cannot open file %s\n", argv[1] );
22         exit ( 1 );
23     }
24
25     /* read image */
26     if ( read_TIFF ( fp, &input_img ) ) {
27         fprintf ( stderr, "error reading file %s\n", argv[1] );
28         exit ( 1 );
29     }
30
31     /* close image file */
32     fclose ( fp );
33     get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
34
35     double*** inter_img = (double***)calloc(3, sizeof(double**));
36     for(int i = 0; i < 3; i++)
37     {
38         inter_img[i] = (double**)calloc(input_img.height, sizeof(double*));
39         for(int j = 0; j < input_img.height; j++)
40         {
41             inter_img[i][j] = (double*)calloc(input_img.width, sizeof(double));
42         }
43     }
44     for(int i = 0; i < input_img.height; i++)
45     {
46         for(int j = 0; j < input_img.width; j++)
47         {
48             inter_img[0][i][j] = 0.01 * input_img.color[0][i][j];
49             inter_img[1][i][j] = 0.01 * input_img.color[1][i][j];
50             inter_img[2][i][j] = 0.01 * input_img.color[2][i][j];
51
52             if(i > 0)
53             {
54                 inter_img[0][i][j] += 0.9 * inter_img[0][i - 1][j];
55                 inter_img[1][i][j] += 0.9 * inter_img[1][i - 1][j];
56                 inter_img[2][i][j] += 0.9 * inter_img[2][i - 1][j];
57             }
58             if(j > 0)
59             {
60                 inter_img[0][i][j] += 0.9 * inter_img[0][i][j - 1];
61                 inter_img[1][i][j] += 0.9 * inter_img[1][i][j - 1];
62                 inter_img[2][i][j] += 0.9 * inter_img[2][i][j - 1];
63             }
64             if(i > 0 && j > 0)
65             {
66                 inter_img[0][i][j] += 0.9 * inter_img[0][i - 1][j - 1];
67                 inter_img[1][i][j] += 0.9 * inter_img[1][i - 1][j - 1];
68                 inter_img[2][i][j] += 0.9 * inter_img[2][i - 1][j - 1];
69             }
70         }
71     }
72 }
```

```

66         inter_img[0][i][j] -= 0.81 * inter_img[0][i - 1][j -
1];
67         inter_img[1][i][j] -= 0.81 * inter_img[1][i - 1][j -
1];
68         inter_img[2][i][j] -= 0.81 * inter_img[2][i - 1][j -
1];
69     }
70 }
71
72 for(int d = 0; d < 3; d++)
73 {
74     for(int i = 0; i < input_img.height; i++)
75     {
76         for(int j = 0; j < input_img.width; j++)
77         {
78             color_img.color[d][i][j] = limitPixel(inter_img[d][
i][j]);
79         }
80     }
81 }
82
83
84 if ( ( fp = fopen ( "color_IIR.tif", "wb" ) ) == NULL ) {
85     fprintf ( stderr, "cannot open file color.tif\n" );
86     exit ( 1 );
87 }
88
89 /* write color image */
90 if ( write_TIFF ( fp, &color_img ) ) {
91     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
92     exit ( 1 );
93 }
94
95 fclose ( fp );
96
97 free_TIFF ( &(input_img) );
98 free_TIFF ( &(color_img) );
99
100 return 0;
101 }
102
103 void error(char *name)
104 {
105     printf("usage: %s image.tiff \n\n",name);
106     printf("this program reads in a 24-bit color TIFF image.\n");
107     printf("It then horizontally filters the green component, adds
noise,\n");
108     printf("and writes out the result as an 8-bit image\n");
109     printf("with the name 'green.tiff'.\n");
110     printf("It also generates an 8-bit color image,\n");
111     printf("that swaps red and green components from the input
image");
112     exit(1);
113 }
```

## 6 Python code

### 6.1 Problem3

```
1 import sys
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import scipy.fft
5 import scipy.signal
6 from matplotlib import cm
7
8 h = (1/81) * np.ones((9,9))
9 fft_out = np.fft.fft2(h, s=[1024,1024])
10 fft_out = np.fft.fftshift(fft_out)
11
12 #plt.plot(np.linspace(-512, 512, 1024), abs(fft_out[1]))
13 #plt.show()
14
15 x = np.linspace(-np.pi, np.pi, 1024)
16 y = np.linspace(-np.pi, np.pi, 1024)
17 X, Y = np.meshgrid(x,y)
18 Z = abs(fft_out)
19 fig = plt.figure()
20 ax = plt.axes(projection='3d')
21 ax.contour3D(X, Y, Z, 50, cmap='viridis')
22 ax.set_xlabel('x')
23 ax.set_ylabel('y')
24 ax.set_zlabel('z')
25 ax.set_title("Magnitude of H")
26 #plt.show()
27
28 ##plt.plot(np.linspace(-np.pi, np.pi, 1024), np.angle(fft_out[1]))
29 #plt.show()
30
31 x = np.linspace(-np.pi, np.pi, 1024)
32 y = np.linspace(-np.pi, np.pi, 1024)
33 X, Y = np.meshgrid(x,y)
34 Z = np.angle(fft_out)
35 fig = plt.figure()
36 ax = plt.axes(projection='3d')
37 ax.contour3D(X, Y, Z, 50, cmap='viridis')
38 ax.set_xlabel('x')
39 ax.set_ylabel('y')
40 ax.set_zlabel('z')
41 ax.set_title("Phase of H")
42 plt.show()
```

### 6.2 Problem4

```
1 import sys
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import scipy.fft
5 import scipy.signal
6 from matplotlib import cm
7
```

```

8  h = (1/25) * np.ones((5,5))
9
10 fft_out = np.fft.fft2(h, s=[1024,1024])
11 fft_out = np.fft.fftshift(fft_out)
12
13 #plt.plot(np.linspace(-512, 512, 1024), abs(fft_out[1]))
14 #plt.show()
15
16 x = np.linspace(-512, 512, 1024)
17 y = np.linspace(-512, 512, 1024)
18 X, Y = np.meshgrid(x,y)
19 Z = abs(fft_out)
20 fig = plt.figure(1)
21 ax = plt.axes(projection='3d')
22 ax.contour3D(X, Y, Z, 50, cmap='viridis')
23 ax.set_xlabel('x')
24 ax.set_ylabel('y')
25 ax.set_zlabel('z')
26 ax.set_title("Magnitude of H")
27 #plt.show()
28
29 #plt.plot(np.linspace(-np.pi, np.pi, 1024), np.angle(fft_out[1]))
30 #plt.show()
31
32 x = np.linspace(-np.pi, np.pi, 1024)
33 y = np.linspace(-np.pi, np.pi, 1024)
34 X, Y = np.meshgrid(x,y)
35 Z = np.angle(fft_out)
36 fig = plt.figure(2)
37 ax = plt.axes(projection='3d')
38 ax.contour3D(X, Y, Z, 50, cmap='viridis')
39 ax.set_xlabel('x')
40 ax.set_ylabel('y')
41 ax.set_zlabel('z')
42 ax.set_title("Phase of H")
43 #plt.show()
44
45
46 g = np.zeros((5,5))
47 lambda_var = 1.5
48 delta = 0;
49 for i in range(0,5):
50     for j in range(0,5):
51         if(i == 2 and j == 2):
52             delta = 1
53         else:
54             delta = 0
55         g[i][j] = delta + lambda_var*(delta - h[i][j])
56
57 fft_out = np.fft.fft2(g, s=[1024,1024])
58 fft_out = np.fft.fftshift(fft_out)
59
60 #plt.plot(np.linspace(-512, 512, 1024), abs(fft_out[1]))
61 #plt.show()
62
63 x = np.linspace(-512, 512, 1024)
64 y = np.linspace(-512, 512, 1024)

```

```

65 X, Y = np.meshgrid(x,y)
66 Z = abs(fft_out)
67 fig = plt.figure(3)
68 ax = plt.axes(projection='3d')
69 ax.contour3D(X, Y, Z, 50, cmap='viridis')
70 ax.set_xlabel('x')
71 ax.set_ylabel('y')
72 ax.set_zlabel('z')
73 ax.set_title("Magnitude of G")
74 #plt.show()
75
76 #plt.plot(np.linspace(-np.pi, np.pi, 1024), np.angle(fft_out[1]))
77 #plt.show()
78
79 x = np.linspace(-np.pi, np.pi, 1024)
80 y = np.linspace(-np.pi, np.pi, 1024)
81 X, Y = np.meshgrid(x,y)
82 Z = np.angle(fft_out)
83 fig = plt.figure(4)
84 ax = plt.axes(projection='3d')
85 ax.contour3D(X, Y, Z, 50, cmap='viridis')
86 ax.set_xlabel('x')
87 ax.set_ylabel('y')
88 ax.set_zlabel('z')
89 ax.set_title("Phase of G")
90 plt.show()

```

### 6.3 Problem5

```

1 import sys
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from PIL import Image
5 import math
6 import cmath
7
8 input_array = np.zeros((256,256))
9 input_array[127][127] = 1
10 output_array = np.zeros((256,256))
11
12 for i in range(0,256):
13     for j in range(0,256):
14         output_array[i][j] = 0.01 * input_array[i][j]
15         if(i > 0):
16             output_array[i][j] += 0.9 * output_array[i - 1][j]
17             if(j > 0):
18                 output_array[i][j] += 0.9 * output_array[i][j - 1]
19             if(i > 0 and j > 0):
20                 output_array[i][j] -= 0.81 * output_array[i - 1][j - 1]
21
22 im_save = Image.fromarray((255 * 100 * output_array).astype(np.
23     uint8))
24 im_save.save("h_out.tif")
25
26 x = np.linspace(-np.pi, np.pi, 1024)
27 y = np.linspace(-np.pi, np.pi, 1024)

```

```

28
29 def genComplexArray(m,n):
30     return (np.arange(m) * 0j)[:, None] + np.arange(n)
31 z = genComplexArray(1024, 1024)
32 X, Y = np.meshgrid(x,y)
33
34 for m in range(0,len(x)) :
35     for n in range(0,len(y)) :
36         z[m][n] = 1 / (1 - (0.9 * cmath.exp(complex(0,-x[m])))) -
37             (0.9 * cmath.exp(complex(0,-y[n]))) + 0.81 * (cmath.exp(complex
38             (0,-(x[m] + y[n]))))
39 Z = abs(z)
40 fig = plt.figure(1)
41 ax = plt.axes(projection='3d')
42 ax.contour3D(X, Y, Z, 50, cmap='viridis')
43 ax.set_xlabel('x')
44 ax.set_ylabel('y')
45 ax.set_zlabel('z')
46 ax.set_title("Magnitude of H")
47 Z = np.angle(z)
48 fig = plt.figure(2)
49 ax = plt.axes(projection='3d')
50 ax.contour3D(X, Y, Z, 50, cmap='viridis')
51 ax.set_xlabel('x')
52 ax.set_ylabel('y')
53 ax.set_zlabel('z')
54 ax.set_title("Phase of H")
55 plt.show()

```