# Digital Image Processing Laboratory: Image Filtering

**Alexandre Olivé Pellicer**

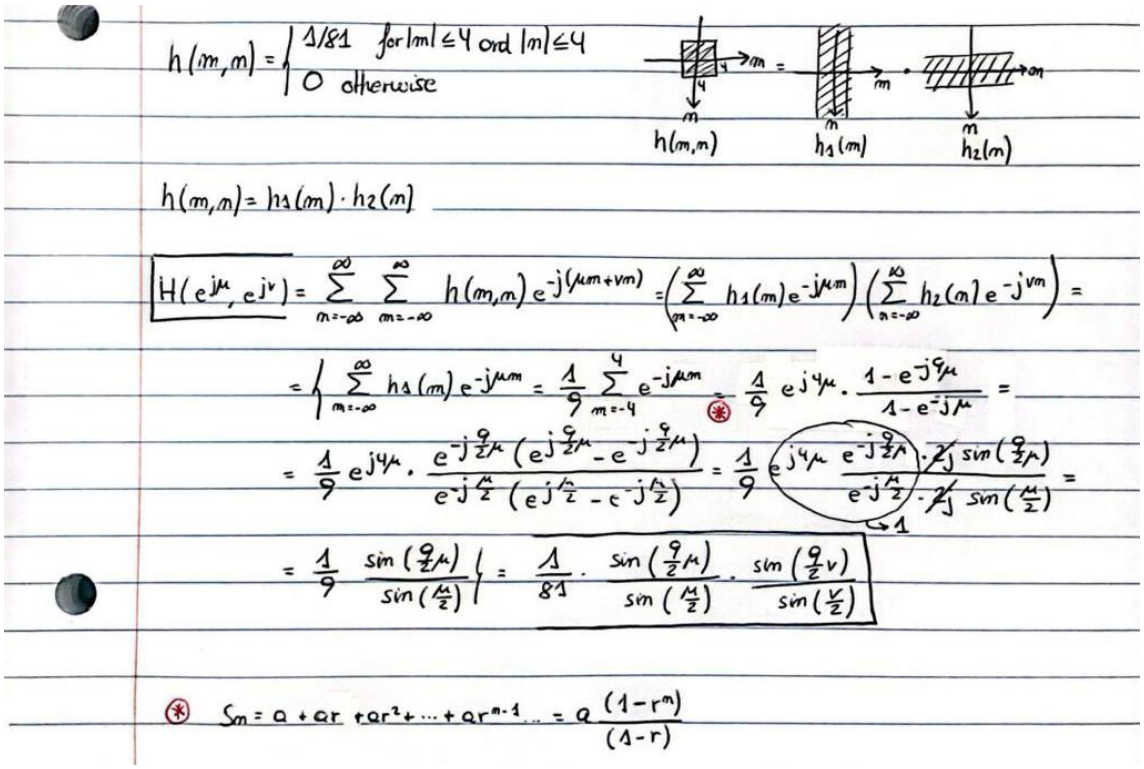
**1/19/2024**

# 1. C Programing

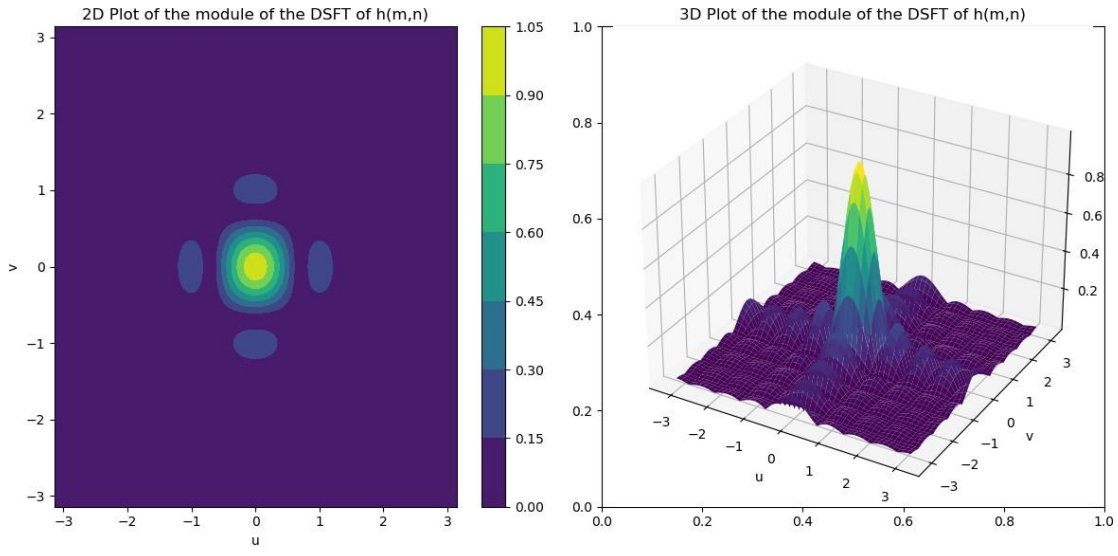
# 2. Displaying and Exporting Images in Python

# 3. FIR Low Pass Filter

3.1 A derivation of the analytical expression for $H(e^{j\mu}, e^{jv})$

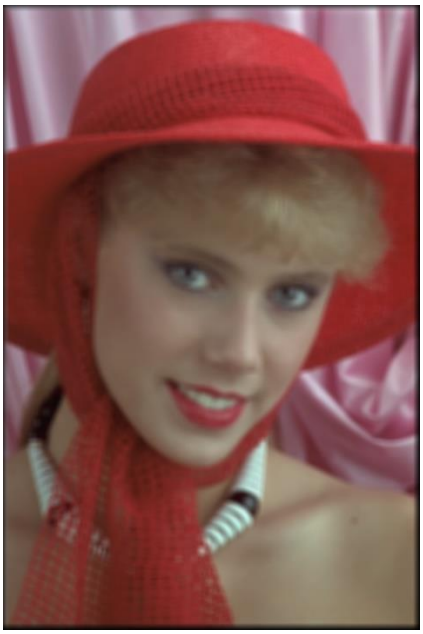



3.2 A plot of $|H(e^{j\mu}, e^{jv})|$



3.3 The color image in *img03.tif*

3.4 The filtered color image



3.5 A listing of my C code

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
```

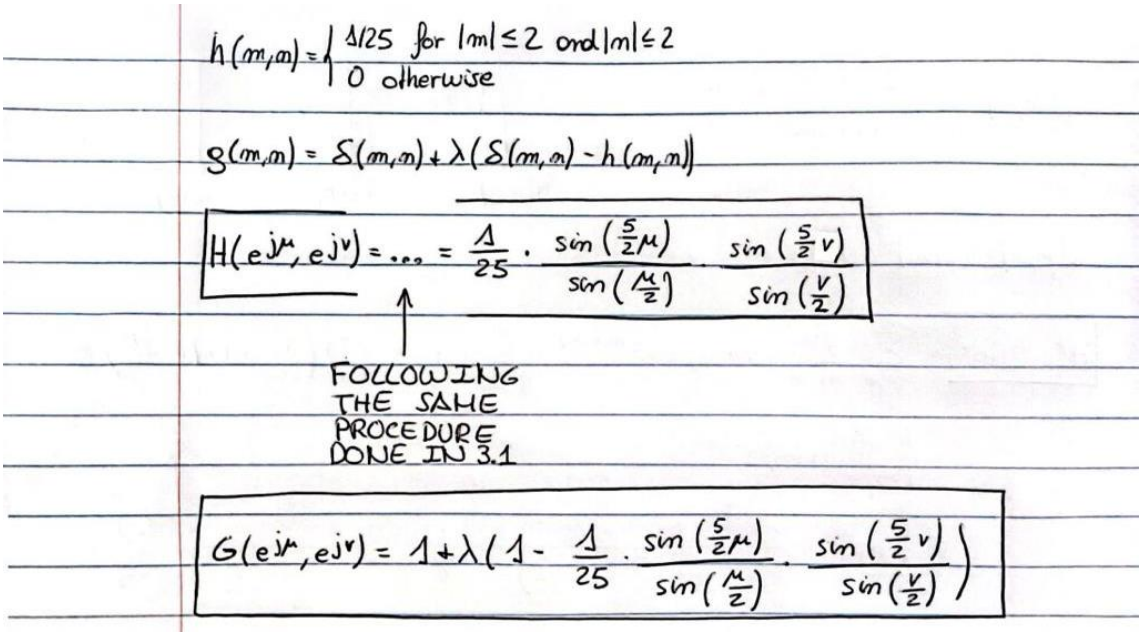
```
void error(char *name);

int main(int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    double **matrix_r, **matrix_g, **matrix_b;
    int32_t i,j,n,m;
    double pix_r, pix_g, pix_b;

    if ( argc != 2 ) error( argv[0] );

    /* 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, &input_img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }

    /* close image file */
    fclose( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
        fprintf ( stderr, "error:  image must be 24-bit color\n" );
        exit ( 1 );
    }

    /* Allocate image of double precision floats */
    matrix_r = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));
    matrix_g = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));
    matrix_b = (double
**)get_img(input_img.width+8,input_img.height+8,sizeof(double));

    get_TIFF( &output_img, input_img.height, input_img.width, 'c' );

    /* Copy rgb component to double array and expand the bound with 0 */
    for (i = 0; i < input_img.height+8; i++) {
        for (j = 0; j < input_img.width+8; j++) {
            matrix_r[i][j] = 0;
            matrix_g[i][j] = 0;
```

```
            matrix_b[i][j] = 0;
        }
    }
    for ( i = 4; i < input_img.height; i++ ){
        for ( j = 4; j < input_img.width; j++ ) {
            matrix_r[i][j] = input_img.color[0][i-4][j-4];
            matrix_g[i][j] = input_img.color[1][i-4][j-4];
            matrix_b[i][j] = input_img.color[2][i-4][j-4];
        }
    }

    /* Filter the image */
    for ( i = 0; i < input_img.height; i++){
        for ( j = 0; j < input_img.width; j++){
            pix_r = 0;
            pix_g = 0;
            pix_b = 0;
            for (m = -4; m<= 4; m++){
                for ( n = -4; n <=4; n++){
                    pix_r = pix_r + (matrix_r[i+4-m][j+4-n])/81;
                    pix_g = pix_g + (matrix_g[i+4-m][j+4-n])/81;
                    pix_b = pix_b + (matrix_b[i+4-m][j+4-n])/81;
                }
            }

            /* Clipping between 0 and 255 */
            output_img.color[0][i][j] = (int)fmin(255, fmax(0, pix_r));
            output_img.color[1][i][j] = (int)fmin(255, fmax(0, pix_g));
            output_img.color[2][i][j] = (int)fmin(255, fmax(0, pix_b));
        }
    }


    /* open output image file */
    if ( ( fp = fopen ( "section3_v2.tif", "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file section3.tif\n");
        exit( 1 );
    }

    /* write output image */
    if ( write_TIFF( fp, &output_img ) ) {
        fprintf( stderr, "error writing TIFF file section3.tif\n");
        exit( 1 );
    }

    /* close output image file */
    fclose( fp );

    /* de-allocate memory */
```

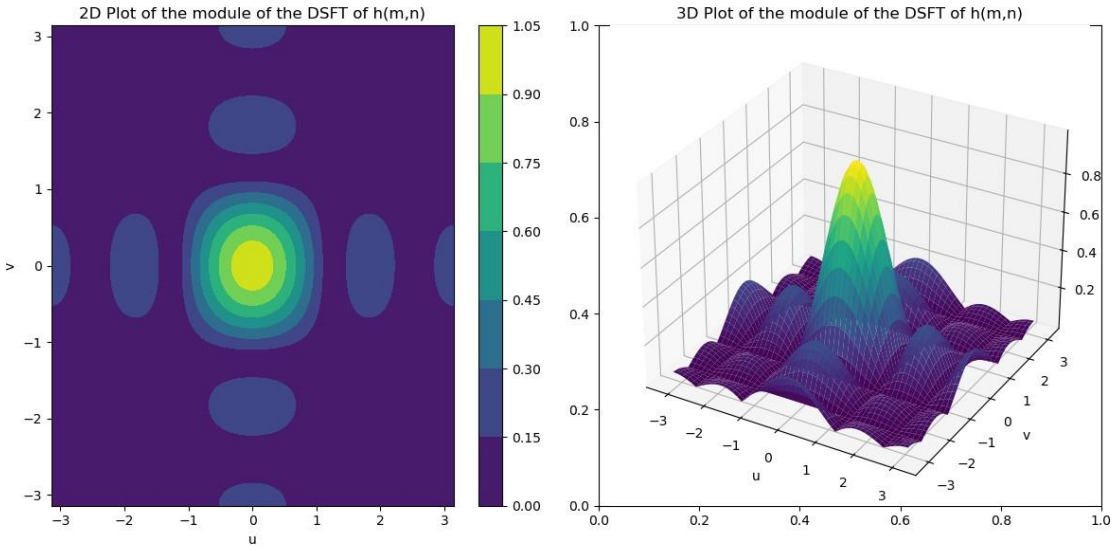
```
    free_TIFF( &(input_img) );
    free_TIFF( &(output_img) );

    free_img( (void**)matrix_r );
    free_img( (void**)matrix_g );
    free_img( (void**)matrix_b );

    return(0);
}

void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds
noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```

## 4. FIR Sharpening Filter

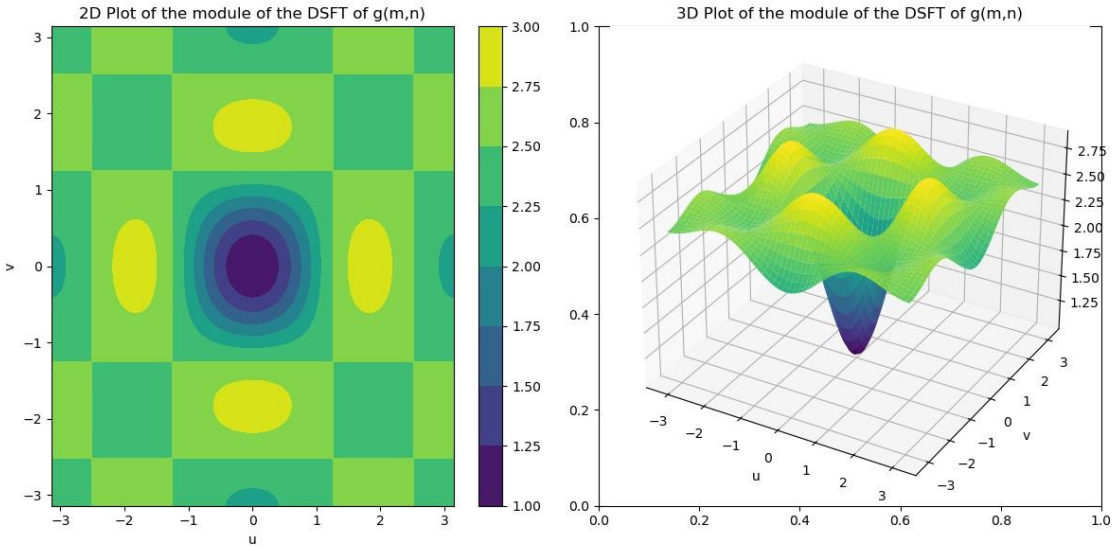
4.1 and 4.2 A derivation of the analytical expression for $H(e^{j\mu}, e^{jv})$ and the analytical expression for $G(e^{j\mu}, e^{jv})$



## 4.3 A plot of $|H(e^{j\mu}, e^{jv})|$




## 4.4 A plot of $|G(e^{j\mu}, e^{jv})|$ for λ = 1.5



4.5 The input color image *imgblur.tif*

4.6 The output sharpened color image for for λ = 1.5

4.7 A listing of my C code

```
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
```

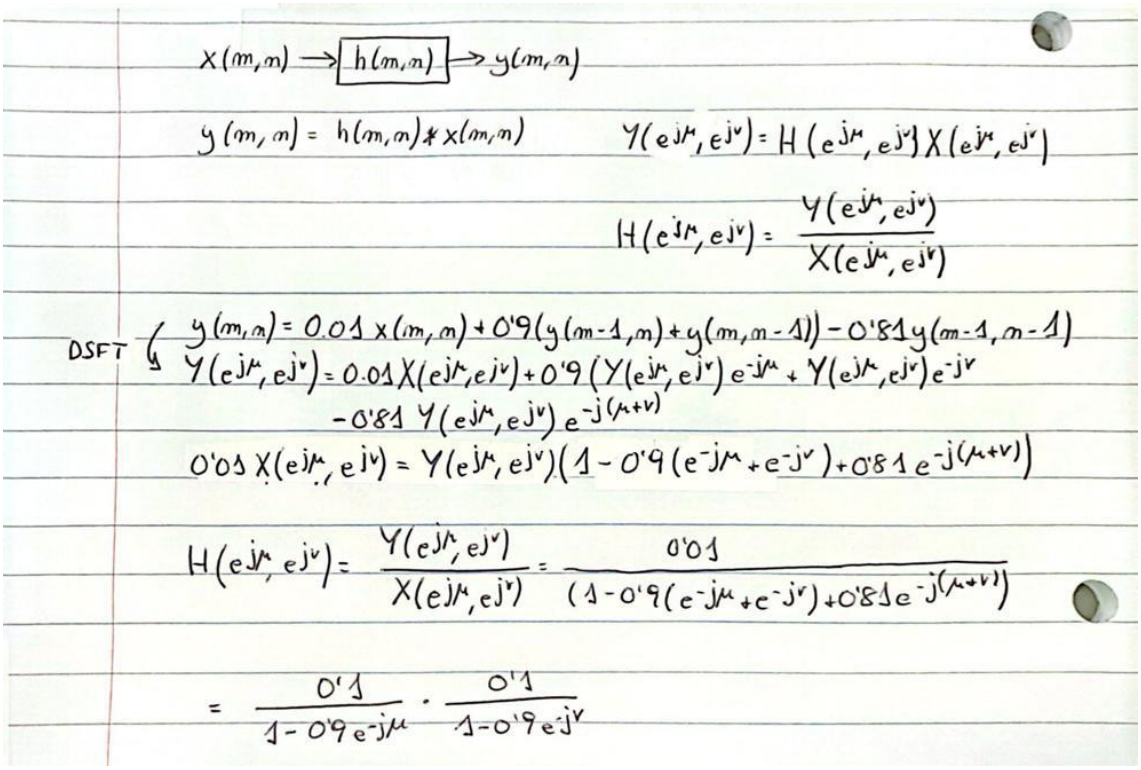
```
void error(char *name);

int main(int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    double **matrix_r, **matrix_g, **matrix_b;
    int32_t i,j,n,m;
    double pix_r, pix_g, pix_b;
    double aux_r, aux_g, aux_b;

    if ( argc != 3 ) error( argv[0] );

    float lambda = atof(argv[2]);

    /* 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, &input_img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }

    /* close image file */
    fclose( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
        fprintf ( stderr, "error:  image must be 24-bit color\n" );
        exit ( 1 );
    }

    /* Allocate image of double precision floats */
    matrix_r = (double **)get_img(input_img.width+4, input_img.height+4,
sizeof(double));
    matrix_g = (double **)get_img(input_img.width+4, input_img.height+4,
sizeof(double));
    matrix_b = (double **)get_img(input_img.width+4, input_img.height+4,
sizeof(double));

    get_TIFF( &output_img, input_img.height, input_img.width, 'c' );

    /* Copy rgb component to double array and expand the bound with 0 */
    for (i = 0; i < input_img.height+4; i++) {
        for (j = 0; j < input_img.width+4; j++) {
```

```
            matrix_r[i][j] = 0;
            matrix_g[i][j] = 0;
            matrix_b[i][j] = 0;
        }
    }
    for ( i = 2; i < input_img.height; i++ ){
        for ( j = 2; j < input_img.width; j++ ) {
            matrix_r[i][j] = input_img.color[0][i-2][j-2];
            matrix_g[i][j] = input_img.color[1][i-2][j-2];
            matrix_b[i][j] = input_img.color[2][i-2][j-2];
        }
    }

    /* Filter the image */
    for ( i = 0; i < input_img.height; i++){
        for ( j = 0; j < input_img.width; j++){
            pix_r = 0;
            pix_g = 0;
            pix_b = 0;
            for (m = -2; m<= 2; m++){
                for ( n = -2; n <=2; n++){
                    pix_r = pix_r + (matrix_r[i+2-m][j+2-n])/25;
                    pix_g = pix_g + (matrix_g[i+2-m][j+2-n])/25;
                    pix_b = pix_b + (matrix_b[i+2-m][j+2-n])/25;
                }
            }

            aux_r = matrix_r[i+2][j+2] + lambda*matrix_r[i+2][j+2]-
lambda*pix_r;
            aux_g = matrix_g[i+2][j+2] + lambda*matrix_g[i+2][j+2]-
lambda*pix_g;
            aux_b = matrix_b[i+2][j+2] + lambda*matrix_b[i+2][j+2]-
lambda*pix_b;

            /* Clipping between 0 and 255 */
            output_img.color[0][i][j] = (int)fmin(255, fmax(0, aux_r));
            output_img.color[1][i][j] = (int)fmin(255, fmax(0, aux_g));
            output_img.color[2][i][j] = (int)fmin(255, fmax(0, aux_b));
        }
    }

    /* open output image file */
    if ( ( fp = fopen ( "section4.tif", "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file section4.tif\n");
        exit( 1 );
    }

    /* write output image */
    if ( write_TIFF( fp, &output_img ) ) {
```

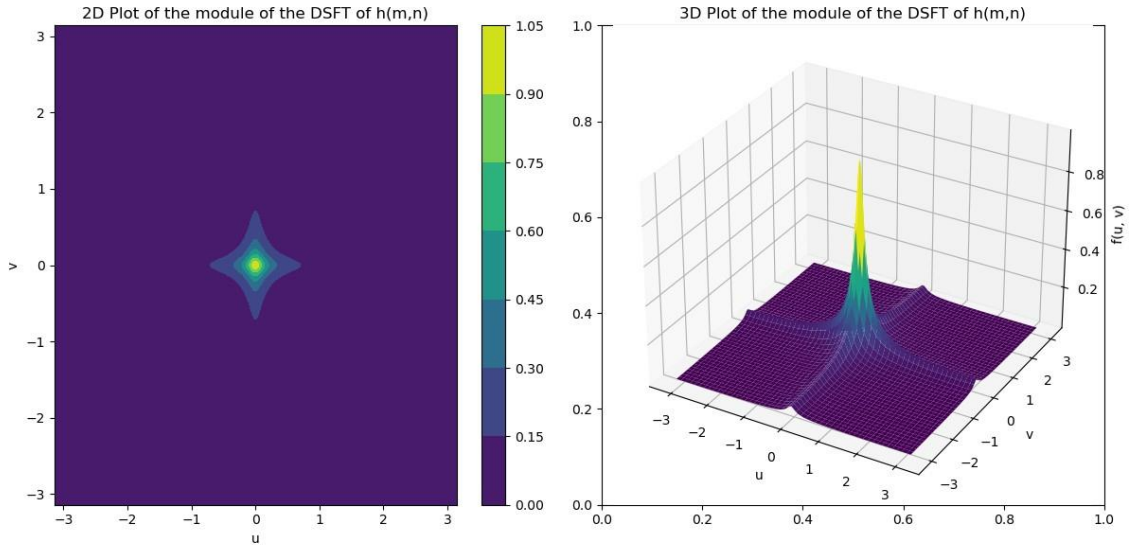
```
        fprintf( stderr, "error writing TIFF file section4.tif\n");
        exit( 1 );
    }

    /* close output image file */
    fclose( fp );

    /* de-allocate memory */
    free_TIFF( &(input_img) );
    free_TIFF( &(output_img) );

    free_img( (void**)matrix_r );
    free_img( (void**)matrix_g );
    free_img( (void**)matrix_b );

    return(0);
 }

 void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds
noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```

# 5. IIR Filter

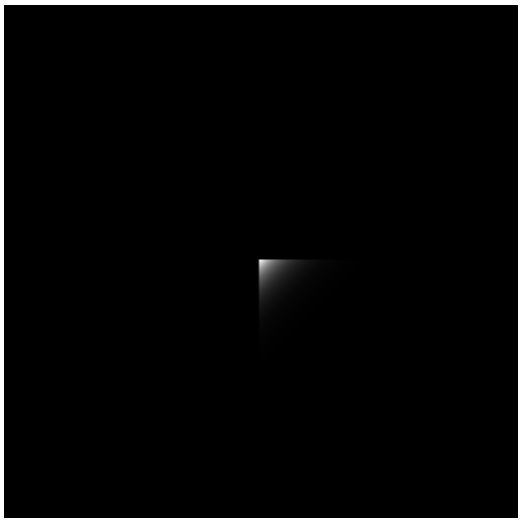
5.1 A derivation of the analytical expression for $H(e^{j\mu}, e^{jv})$




5.2 A plot of $|H(e^{j\mu}, e^{jv})|$



5.3 An image of the point spread function.



5.4 The filtered output color image.

5.5 A listing of my C code

```c
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
```

```
int main(int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, output_img;
    double **matrix1_r, **matrix1_g, **matrix1_b, **matrix2_r,
**matrix2_g, **matrix2_b;
    int32_t i,j;

    if ( argc != 2 ) error( argv[0] );

    /* 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, &input_img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }

    /* close image file */
    fclose( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
        fprintf ( stderr, "error:  image must be 24-bit color\n" );
        exit ( 1 );
    }

    /* Allocate image of double precision floats */
    matrix1_r = (double **)get_img(input_img.width+2, input_img.height+2,
sizeof(double));
    matrix1_g = (double **)get_img(input_img.width+2, input_img.height+2,
sizeof(double));
    matrix1_b = (double **)get_img(input_img.width+2, input_img.height+2,
sizeof(double));
    matrix2_r = (double **)get_img(input_img.width+2, input_img.height+2,
sizeof(double));
    matrix2_g = (double **)get_img(input_img.width+2, input_img.height+2,
sizeof(double));
    matrix2_b = (double **)get_img(input_img.width+2, input_img.height+2,
sizeof(double));

    get_TIFF ( &output_img, input_img.height, input_img.width, 'c' );

    /* Copy rgb component to double array and expand the bound with 0 */
    for ( i = 0; i < input_img.height+2; i++ ){
```

```
        for ( j = 0; j < input_img.width+2; j++ ) {
            matrix1_r[i][j] = 0;
            matrix1_g[i][j] = 0;
            matrix1_b[i][j] = 0;
            matrix2_r[i][j] = 0;
            matrix2_g[i][j] = 0;
            matrix2_b[i][j] = 0;
        }
    }
    for ( i = 1; i < input_img.height; i++ ){
        for ( j = 1; j < input_img.width; j++ ) {
            matrix1_r[i][j] = input_img.color[0][i-1][j-1];
            matrix1_g[i][j] = input_img.color[1][i-1][j-1];
            matrix1_b[i][j] = input_img.color[2][i-1][j-1];
        }
    }

    /* Filter the image */
    for ( i = 0; i < input_img.height; i++ ){
        for ( j = 0; j < input_img.width; j++ ) {
            matrix2_r[i+1][j+1] = 0.01*matrix1_r[i+1][j+1] +
0.9*(matrix2_r[i][j+1]+matrix2_r[i+1][j])-0.81*matrix2_r[i][j];
            matrix2_g[i+1][j+1] = 0.01*matrix1_g[i+1][j+1] +
0.9*(matrix2_g[i][j+1]+matrix2_g[i+1][j])-0.81*matrix2_g[i][j];
            matrix2_b[i+1][j+1] = 0.01*matrix1_b[i+1][j+1] +
0.9*(matrix2_b[i][j+1]+matrix2_b[i+1][j])-0.81*matrix2_b[i][j];

            /* Clipping between 0 and 255 */
            output_img.color[0][i][j] = (int)fmin(255, fmax(0,
matrix2_r[i+1][j+1]));
            output_img.color[1][i][j] = (int)fmin(255, fmax(0,
matrix2_g[i+1][j+1]));
            output_img.color[2][i][j] = (int)fmin(255, fmax(0,
matrix2_b[i+1][j+1]));
        }
    }

    /* open output image file */
    if ( ( fp = fopen ( "section5.tif", "wb" ) ) == NULL ) {
        fprintf( stderr, "cannot open file section5.tif\n");
        exit( 1 );
    }

    /* write output image */
    if ( write_TIFF( fp, &output_img ) ) {
        fprintf( stderr, "error writing TIFF file section5.tif\n");
        exit( 1 );
    }
```

```
    /* close output image file */
    fclose( fp );

    /* de-allocate memory */
    free_TIFF ( &(input_img) );
    free_TIFF ( &(output_img) );

    free_img( (void**)matrix1_r );
    free_img( (void**)matrix1_g );
    free_img( (void**)matrix1_b );
    free_img( (void**)matrix2_r );
    free_img( (void**)matrix2_g );
    free_img( (void**)matrix2_b );

    return(0);
 }

 void error(char *name)
{
    printf("usage:  %s  image.tiff \n\n",name);
    printf("this program reads in a 24-bit color TIFF image.\n");
    printf("It then horizontally filters the green component, adds
noise,\n");
    printf("and writes out the result as an 8-bit image\n");
    printf("with the name 'green.tiff'.\n");
    printf("It also generates an 8-bit color image,\n");
    printf("that swaps red and green components from the input image");
    exit(1);
}
```