**Digital Image Processing Laboratory:**

**Image Filtering**

**Alexandre Olivé Pellicer**

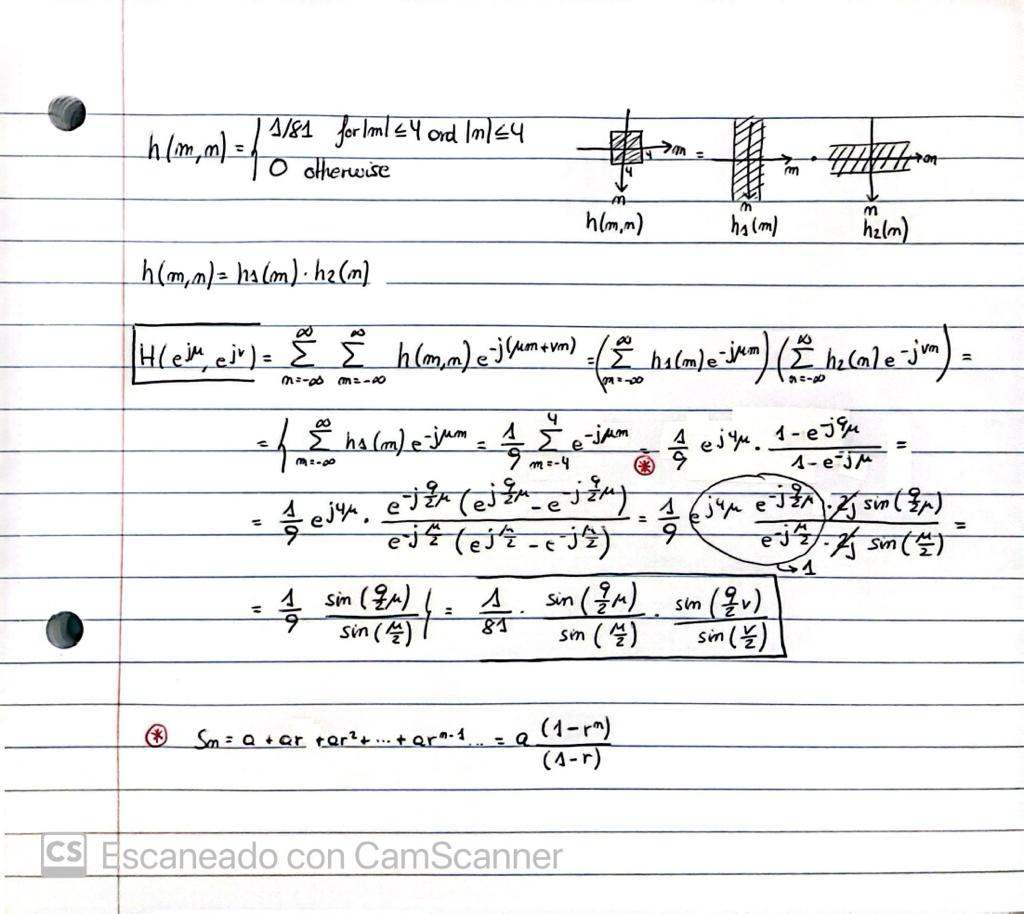
**1/19/2024**

**1. C Programing**

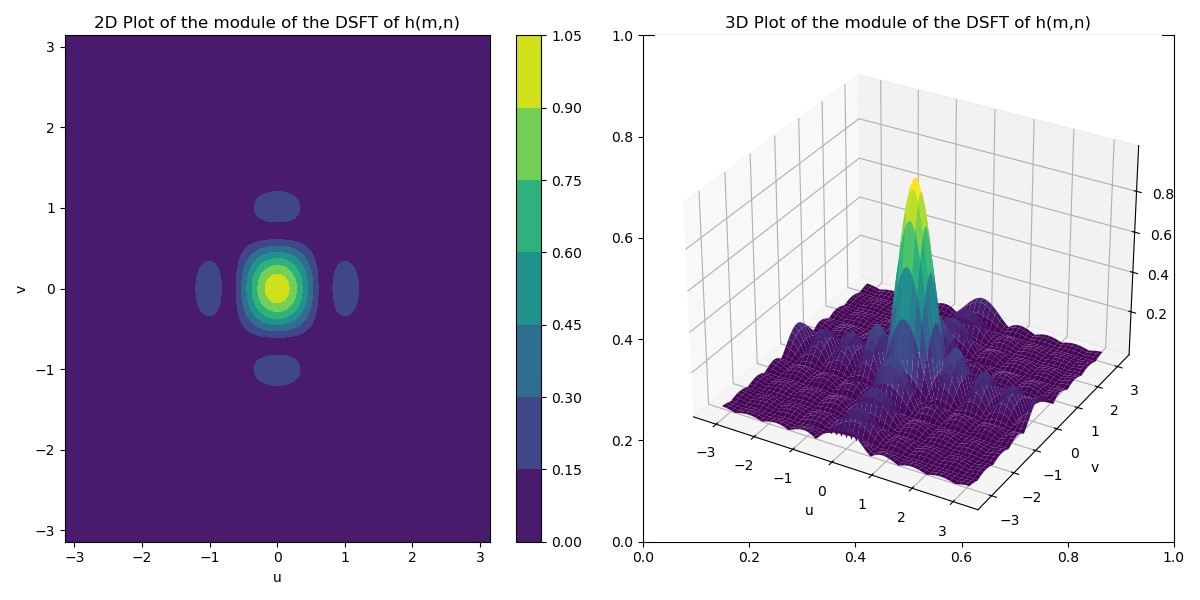
**2. Displaying and Exporting Images in Python**

**3. FIR Low Pass Filter**

3.1 A derivation of the analytical expression for



3.2 A plot of



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");

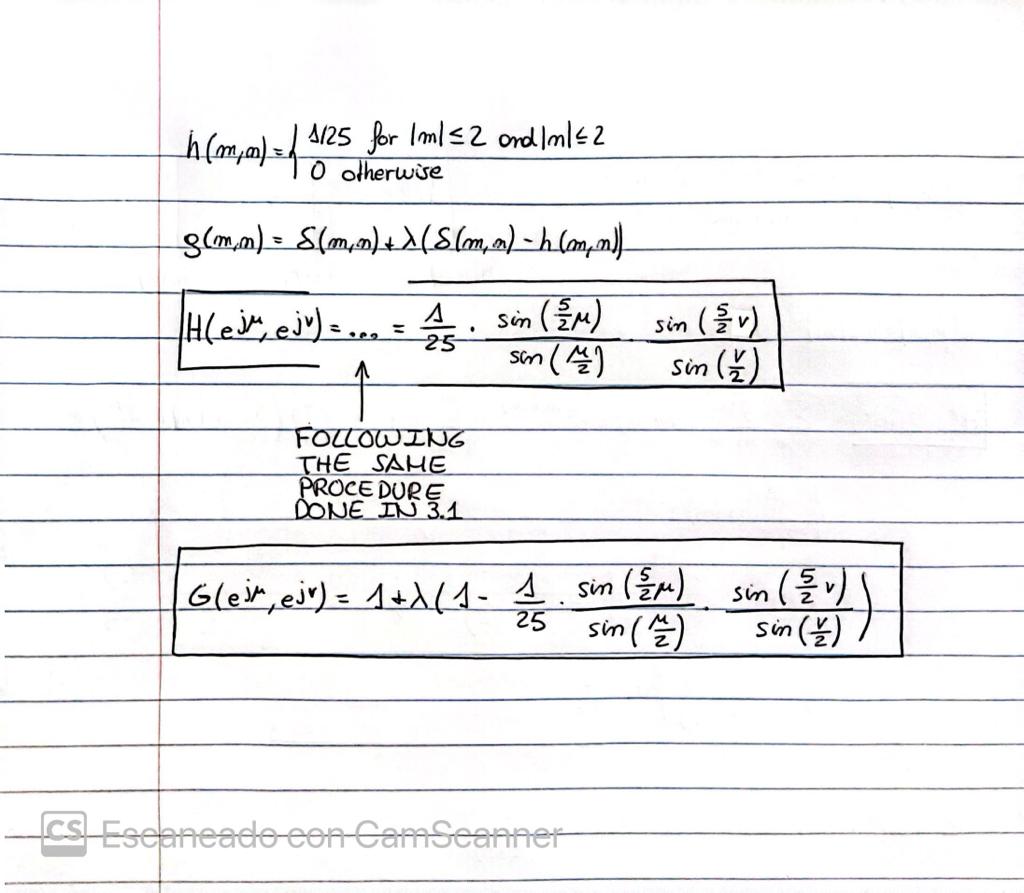
    printf("that swaps red and green components from the input image");

    exit(1);

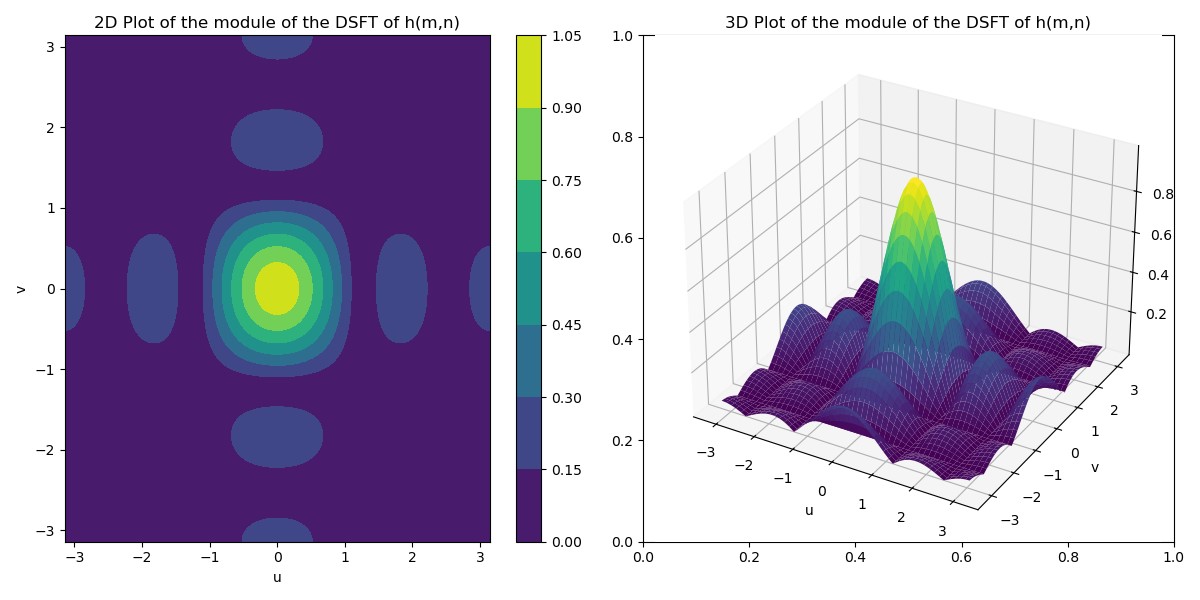
}

**4. FIR Sharpening Filter**

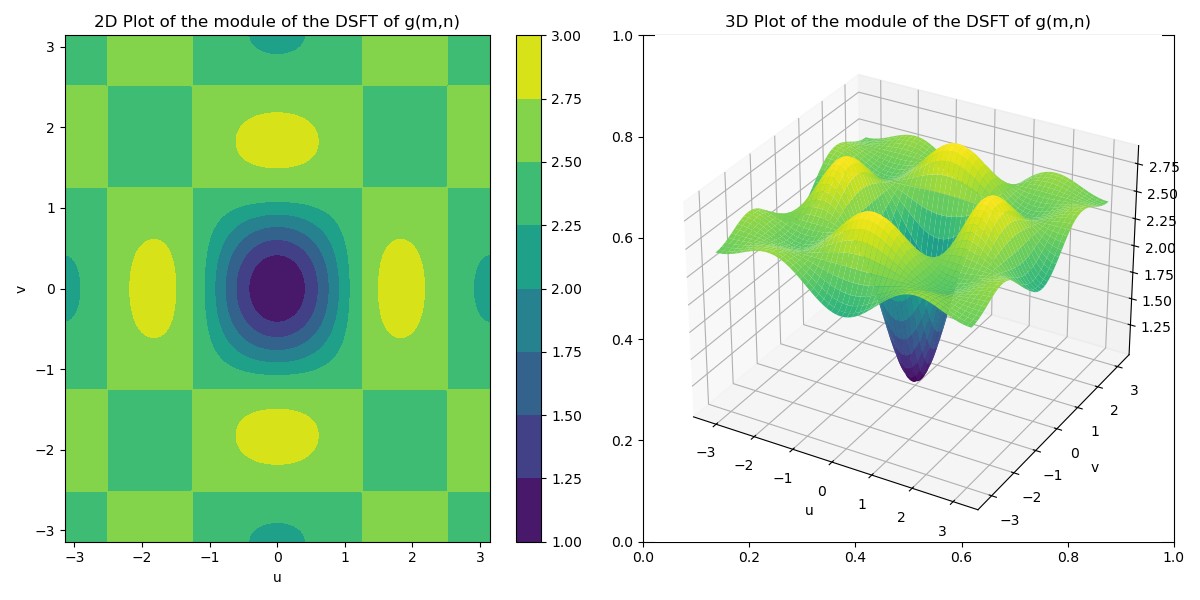
4.1 and 4.2 A derivation of the analytical expression for and the analytical expression for



4.3 A plot of



4.4 A plot of 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");

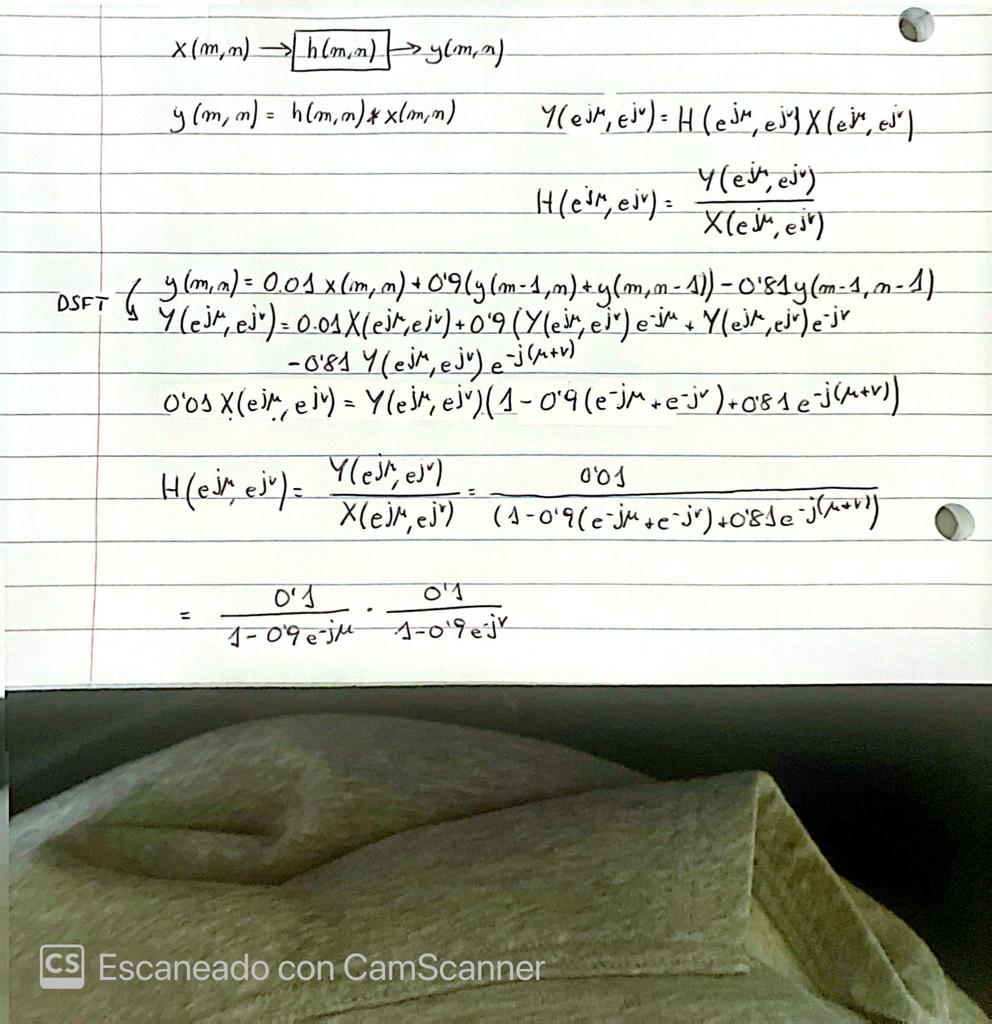
    printf("that swaps red and green components from the input image");

    exit(1);

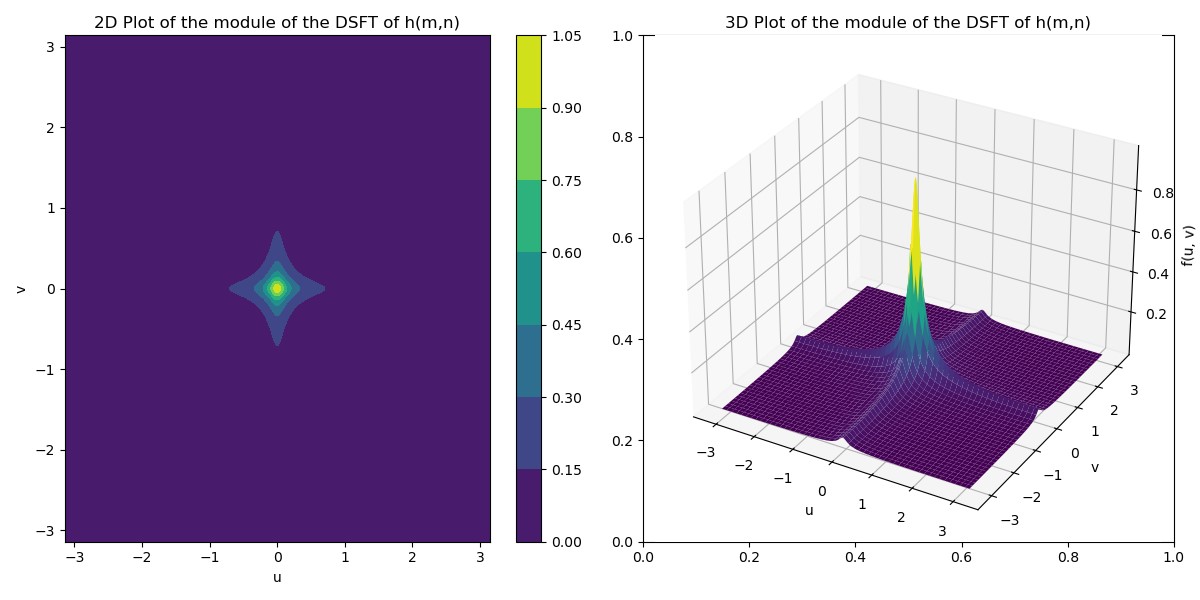
}

**5. IIR Filter**

5.1 A derivation of the analytical expression for



5.2 A plot of



5.3 An image of the point spread function.



5.4 The filtered output color image.



5.5 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 \*\*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);

}