ECE 637 Lab 1 - Image Filtering

Yifan Fei

Electrical Engineering
Purdue University

Instructor: Prof. Charles A. Bouman

Spring 2018

Section 3: FIR Low Pass Filter

1:A derivation of the analytical expression for H(u,v)

The difference equation:

$$y(m,n) = \sum_{k=-4}^{4} \sum_{l=-4}^{4} h(k,l)x(m-k,n-l)$$

the DSFT of h(m,n):

$$H(z_1,z_2) = \sum_{k=-4}^{4} \sum_{l=-4}^{4} h(k,l)z_1^{-k}z_2^{-l}$$

$$H(e^{ju},e^{jv}) = \sum_{k=-4}^{4} \sum_{l=-4}^{4} h(k,l)e^{-j(ku+lv)} = \frac{1}{81}\sum_{k=-4}^{4} \sum_{l=-4}^{4} e^{-j(ku+lv)}$$

2: A plot of magnitude of H(u,v)

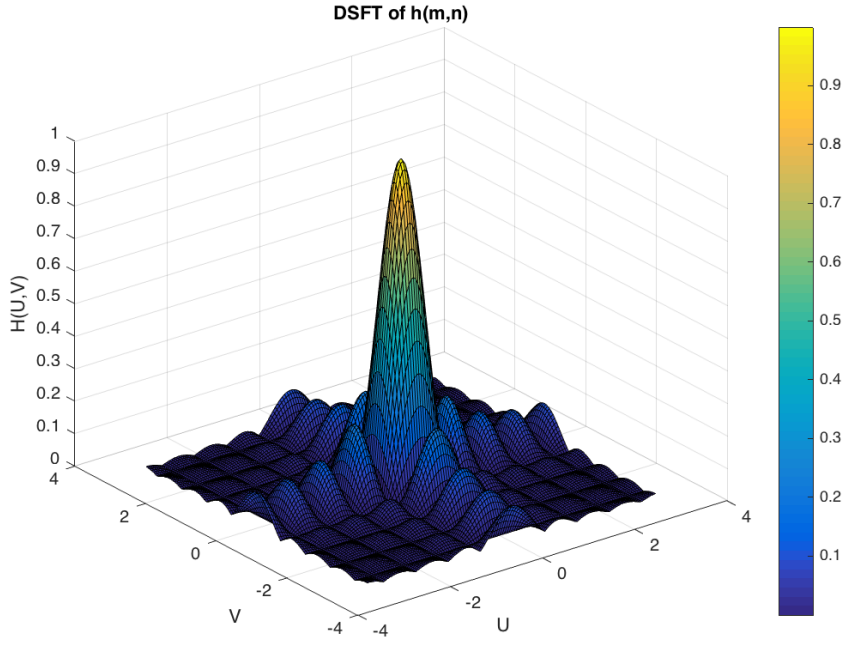


Figure 1: plot of magnitude of H(u,v)

3: The color image in img03.tif

(a) the gray scale image green.tif (b) the color image in img03.tif

Figure 2: pictures by running the Example on img03.tif

4: The filtered color image

(a) the original image img03.tif (b) The filtered color image by FIR Low Pass Filter

Figure 3: pictures by running FIR low pass filter on img03.tif

5: C code

See the attachment

Section 4: FIR Sharpening Filter

1: A derivation of the analytical expression for H(u,v)

The difference equation:

$$y(m,n) = \sum_{k=-2}^{2} \sum_{l=-2}^{2} h(k,l)x(m-k,n-l)$$

the DSFT of h(m,n):

$$H(z_1,z_2) = \sum_{k=-2}^{2} \sum_{l=-2}^{2} h(k,l)z_1^{-k}z_2^{-l}$$

$$H(e^{ju},e^{jv}) = \sum_{k=-2}^{2} \sum_{l=-2}^{2} h(k,l)e^{-j(ku+lv)} = \frac{1}{25} \sum_{k=-2}^{2} \sum_{l=-2}^{2} e^{-j(ku+lv)}$$

2: A derivation of the analytical expression for G(u,v)

The difference equation:

$$g(m,n) = (1+\lambda)\delta(m,n) - \lambda h(m,n)$$

the DSFT of g(m,n):

$$G(e^{ju},e^{jv}) = CSFT\{(1+\lambda)\delta(m,n) - \lambda h(m,n)\} = (1+\lambda) - \lambda H(e^{ju},e^{jv}) = (1+\lambda) - \lambda \frac{1}{25} \sum_{k=-2}^{2} \sum_{l=-2}^{2} e^{-j(ku+lv)}$$

3: A plot of magnitude of H(u,v)

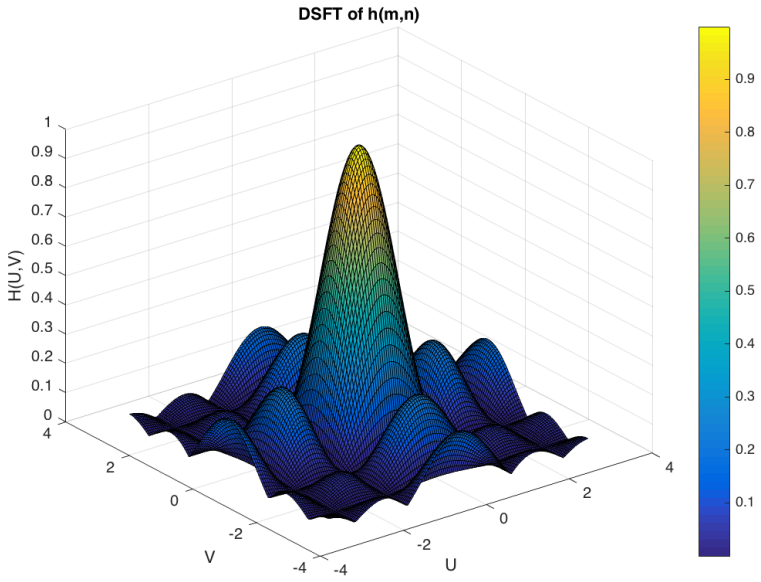


Figure 4: plot of magnitude of H(u,v)

4:A plot of magnitude of G(u,v), lamada = 1.5

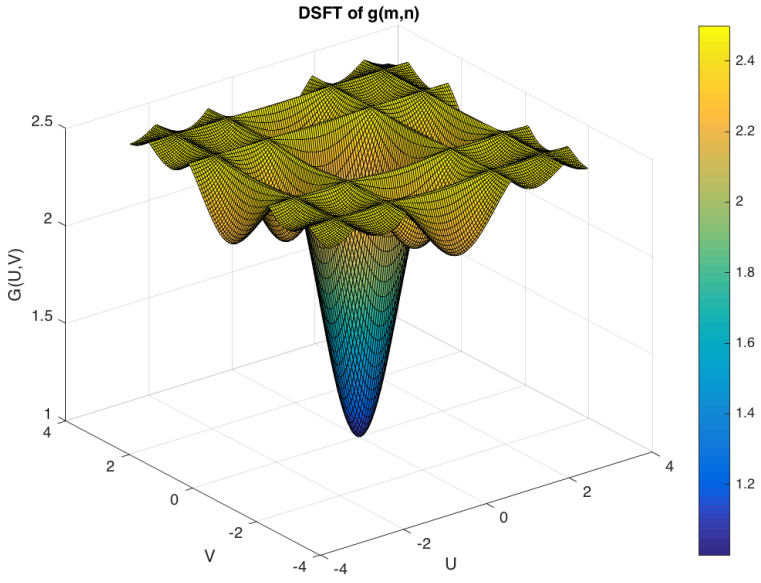
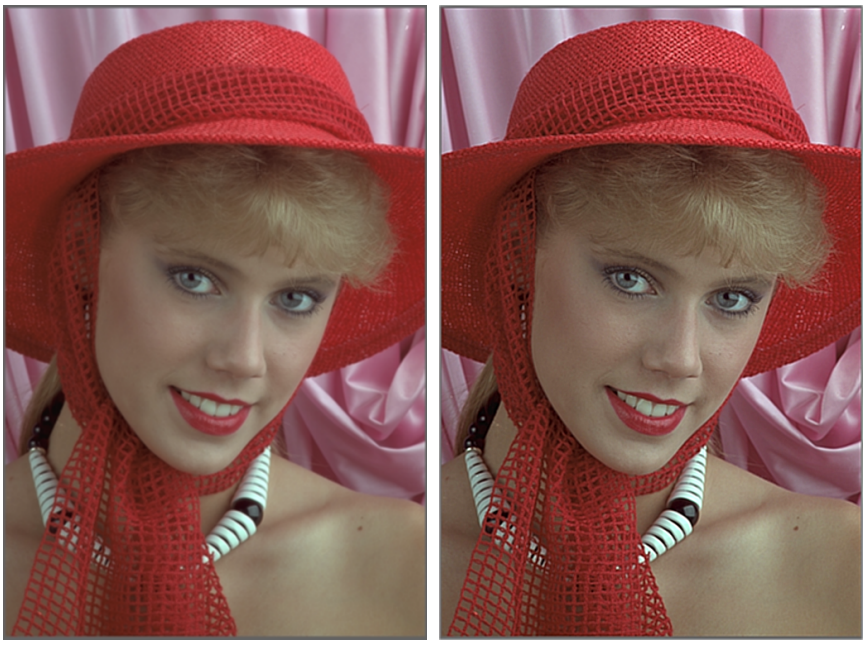


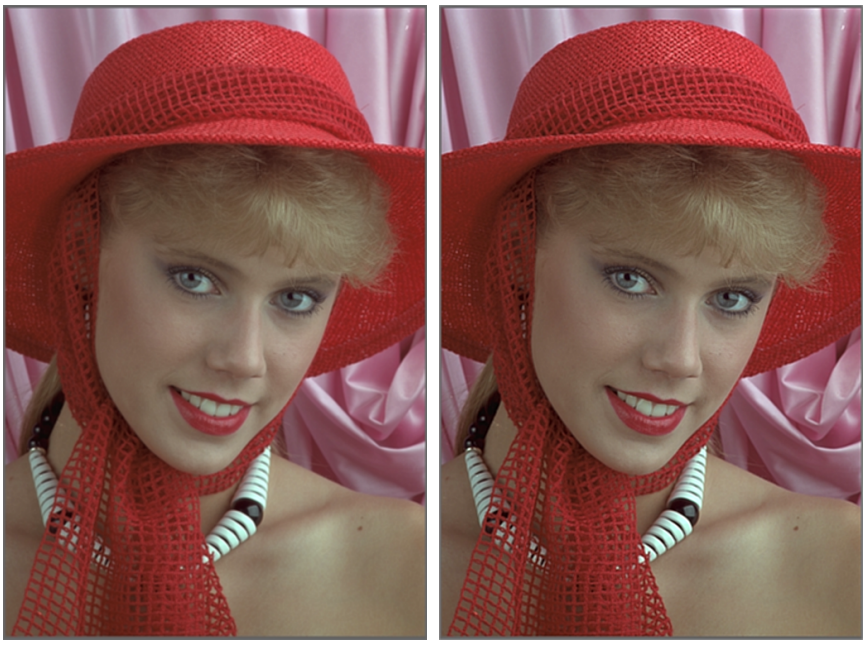
Figure 5: plot of magnitude of G(u,v), lamada = 1.5

5 and 6: The input and output color images



(a) input color image imgblur.tif (b) output sharpened color image for lamada = 1.5

Figure 6: pictures by running the Sharpening filter on imgblur.tif



(a) output sharpened color image for lamada = 0.2 (b) output sharpened color image for lamada = 0.8

Figure 7: pictures by running the Sharpening filter on imgblur.tif

It is shown that as λ increase, the picture becomes much sharper.

7: C code

See attachment.

Section 5: IIR Filter

1:A derivation of the analytical expression for H(u,v)

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

So

$$h(m,n) = \frac{y(m,n)}{x(m,n)} = \frac{0.01}{0.9\delta(m-1) + 0.9\delta(n-1) - 0.81\delta(m-1)\delta(n-1)}$$

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

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

$$H(e^{ju},e^{jv}) = \frac{0.01}{1 - 0.9e^{-ju} - 0.9e^{-jv} + 0.81e^{-ju}e^{-jv}}$$

2:A plot of magnitude of H(u,v)

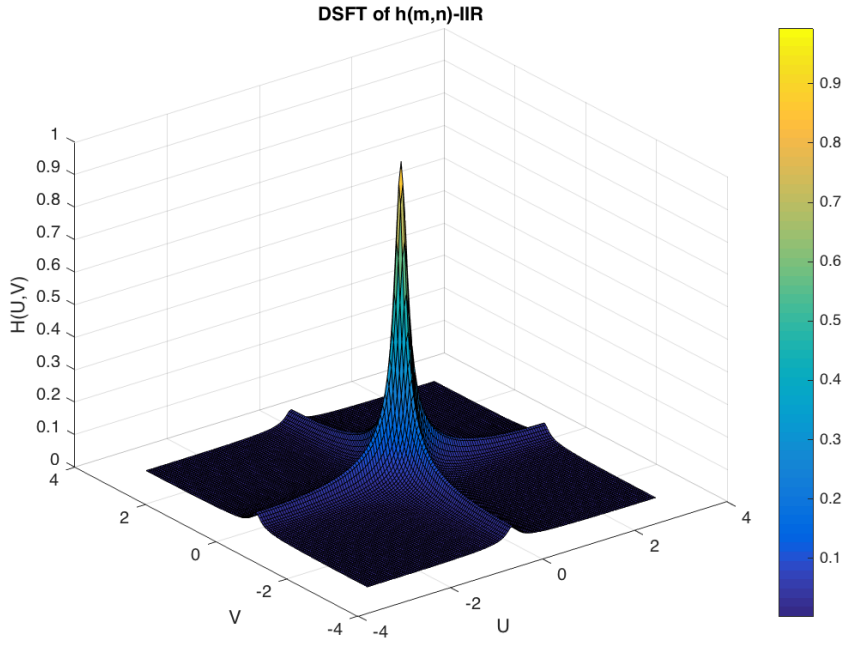


Figure 8: plot of magnitude of H(u,v)

3:An image of the point spread function

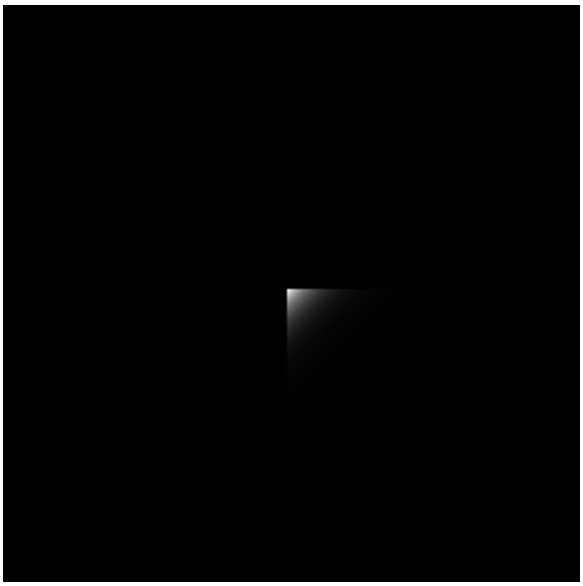
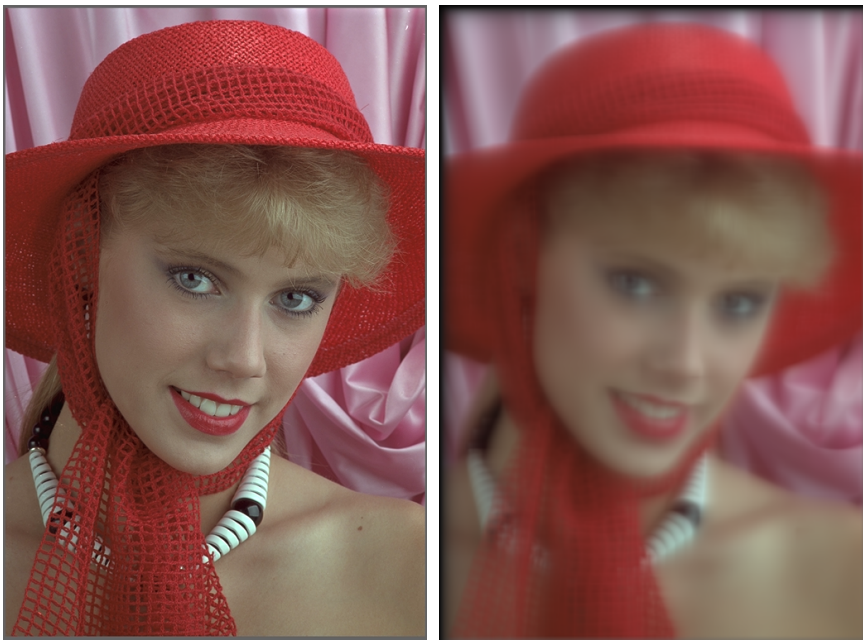


Figure 9: An image of the point spread function

4:The filtered output color image



(a) the original image img03.tif (b) The filtered color image by IIR Filter

Figure 10: pictures by running IIR filter

5:C code

See attachment.

Attachments: code

C code for section 3

```
/* gcc -o FIR_LP FIR_LP.c allocate.c randlib.c solve.c tiff.c */
/* ./FIR_LP img03.tif */

#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, green_img, red_img, blue_img, color_img;
  double **img1,**img2,**img3,**img4,**img5,**img6;
  int32_t i,j,pixel,m,n;

  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 */
  img1 = (double **)get_img(input_img.width+8,input_img.height+8,sizeof(double));
  img2 = (double **)get_img(input_img.width+8,input_img.height+8,sizeof(double));
```

```
img3 = (double **)get_img(input_img.width+8,input_img.height+8,sizeof(double));
img4 = (double **)get_img(input_img.width+8,input_img.height+8,sizeof(double));
img5 = (double **)get_img(input_img.width+8,input_img.height+8,sizeof(double));
img6 = (double **)get_img(input_img.width+8,input_img.height+8,sizeof(double));

/* Initialization of rgb matrix */
for ( i = 0; i < input_img.height+8; i++ )
for ( j = 0; j < input_img.width+8; j++ ) {
  img1[i][j] = 0;
  img3[i][j] = 0;
  img5[i][j] = 0;
  img2[i][j] = 0;
  img4[i][j] = 0;
  img6[i][j] = 0;
}

/* copy rgb component from input to double array */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  img1[i+4][j+4] = input_img.color[0][i][j];
  img3[i+4][j+4] = input_img.color[1][i][j];
  img5[i+4][j+4] = input_img.color[2][i][j];
}


/* FIR Filter */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  for ( m = -4; m <= 4; m++ )
  for ( n = -4; n <= 4; n++ ) {
      img2[i+4][j+4] += img1[i+4+m][j+4+n]/81;
      img4[i+4][j+4] += img3[i+4+m][j+4+n]/81;
      img6[i+4][j+4] += img5[i+4+m][j+4+n]/81;
  }
}

/* set up structure for output achromatic image */
/* to allocate a full color image use type 'c' */
get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );

/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

/* copy rgb component to new images */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  pixel = (int32_t)img2[i+4][j+4];
  if(pixel>255) {
    red_img.mono[i][j] = 255;
  }
  else {
```

```
    if(pixel<0) green_img.mono[i][j] = 0;
    else red_img.mono[i][j] = pixel;
  }

  pixel = (int32_t)img4[i+4][j+4];
  if(pixel>255) {
    green_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) green_img.mono[i][j] = 0;
    else green_img.mono[i][j] = pixel;
  }

  pixel = (int32_t)img6[i+4][j+4];
  if(pixel>255) {
    blue_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) blue_img.mono[i][j] = 0;
    else blue_img.mono[i][j] = pixel;
  }
  color_img.color[0][i][j] = red_img.mono[i][j];
  color_img.color[1][i][j] = green_img.mono[i][j];
  color_img.color[2][i][j] = blue_img.mono[i][j];
}

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

/* write color image */
if ( write_TIFF ( fp, &color_img ) ) {
    fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
    exit ( 1 );
}

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

/* de-allocate space which was used for the images */
free_TIFF ( &(input_img) );
free_TIFF ( &(green_img) );
free_TIFF ( &(red_img) );
free_TIFF ( &(blue_img) );
free_TIFF ( &(color_img) );

free_img( (void**)img1 );
free_img( (void**)img2 );
free_img( (void**)img3 );
free_img( (void**)img4 );
free_img( (void**)img5 );
free_img( (void**)img6 );
```

```
  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);
}
```

C code for section 4

```
/* gcc -o FIR_S FIR_S.c allocate.c randlib.c solve.c tiff.c */
/* ./FIR_S img03.tif 0.2*/

#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, green_img, red_img, blue_img, color_img;
  double **img1,**img2,**img3,**img4,**img5,**img6;
  int32_t i,j,pixel,m,n;
  double lamada = atof(argv[2]);
  printf("lamada: %f", lamada);

  if ( argc != 3 ) 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 */
img1 = (double **)get_img(input_img.width+4,input_img.height+4,sizeof(double));
img2 = (double **)get_img(input_img.width+4,input_img.height+4,sizeof(double));
img3 = (double **)get_img(input_img.width+4,input_img.height+4,sizeof(double));
img4 = (double **)get_img(input_img.width+4,input_img.height+4,sizeof(double));
img5 = (double **)get_img(input_img.width+4,input_img.height+4,sizeof(double));
img6 = (double **)get_img(input_img.width+4,input_img.height+4,sizeof(double));

/* Initialization of rgb matrix */
for ( i = 0; i < input_img.height+4; i++ )
for ( j = 0; j < input_img.width+4; j++ ) {
  img1[i][j] = 0;
  img3[i][j] = 0;
  img5[i][j] = 0;
  img2[i][j] = 0;
  img4[i][j] = 0;
  img6[i][j] = 0;
}

/* copy green component to double array */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  img1[i+2][j+2] = input_img.color[0][i][j];
  img3[i+2][j+2] = input_img.color[1][i][j];
  img5[i+2][j+2] = input_img.color[2][i][j];
}


/* Filter image along horizontal direction */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {

  for ( m = -2; m <= 2; m++ )
  for ( n = -2; n <= 2; n++ ) {
      img2[i+2][j+2] -= lamada*(img1[i+2+m][j+2+n]/25);
      img4[i+2][j+2] -= lamada*(img3[i+2+m][j+2+n]/25);
      img6[i+2][j+2] -= lamada*(img5[i+2+m][j+2+n]/25);
  }
  img2[i+2][j+2] += img1[i+2][j+2]*(1+lamada);
  img4[i+2][j+2] += img3[i+2][j+2]*(1+lamada);
  img6[i+2][j+2] += img5[i+2][j+2]*(1+lamada);
}

/* set up structure for output achromatic image */
/* to allocate a full color image use type 'c' */
get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
```

```
get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );

/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

/* copy rgb component to new images */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  pixel = (int32_t)img2[i+2][j+2];
  if(pixel>255) {
    red_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) green_img.mono[i][j] = 0;
    else red_img.mono[i][j] = pixel;
  }

  pixel = (int32_t)img4[i+2][j+2];
  if(pixel>255) {
    green_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) green_img.mono[i][j] = 0;
    else green_img.mono[i][j] = pixel;
  }

  pixel = (int32_t)img6[i+2][j+2];
  if(pixel>255) {
    blue_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) blue_img.mono[i][j] = 0;
    else blue_img.mono[i][j] = pixel;
  }
  color_img.color[0][i][j] = red_img.mono[i][j];
  color_img.color[1][i][j] = green_img.mono[i][j];
  color_img.color[2][i][j] = blue_img.mono[i][j];
}


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

/* write color image */
if ( write_TIFF ( fp, &color_img ) ) {
    fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
    exit ( 1 );
}

/* close color image file */
```

```
  fclose ( fp );

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(green_img) );
  free_TIFF ( &(red_img) );
  free_TIFF ( &(blue_img) );
  free_TIFF ( &(color_img) );

  free_img( (void**)img1 );
  free_img( (void**)img2 );
  free_img( (void**)img3 );
  free_img( (void**)img4 );
  free_img( (void**)img5 );
  free_img( (void**)img6 );

  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);
}
```

C code for section 5

```
/* gcc -o IIR IIR.c allocate.c randlib.c solve.c tiff.c */
/* ./IIR img03.tif */

#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, green_img, red_img, blue_img, color_img;
  double **img1,**img2,**img3,**img4,**img5,**img6;
  int32_t i,j,pixel,m,n;

  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 */
img1 = (double **)get_img(input_img.width+2,input_img.height+2,sizeof(double));
img2 = (double **)get_img(input_img.width+2,input_img.height+2,sizeof(double));
img3 = (double **)get_img(input_img.width+2,input_img.height+2,sizeof(double));
img4 = (double **)get_img(input_img.width+2,input_img.height+2,sizeof(double));
img5 = (double **)get_img(input_img.width+2,input_img.height+2,sizeof(double));
img6 = (double **)get_img(input_img.width+2,input_img.height+2,sizeof(double));


/* Initialization of rgb matrix */
for ( i = 0; i < input_img.height+2; i++ )
for ( j = 0; j < input_img.width+2; j++ ) {
  img1[i][j] = 0;
  img3[i][j] = 0;
  img5[i][j] = 0;
  img2[i][j] = 0;
  img4[i][j] = 0;
  img6[i][j] = 0;
}

/* copy rgb component to double array */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  img1[i+1][j+1] = input_img.color[0][i][j];
  img3[i+1][j+1] = input_img.color[1][i][j];
  img5[i+1][j+1] = input_img.color[2][i][j];
}


/* Filter image along horizontal direction */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
    img2[i+1][j+1] = 0.01*img1[i+1][j+1] + 0.9*(img2[i][j+1]+img2[i+1][j])-0.81*img2[i][j
    img4[i+1][j+1] = 0.01*img3[i+1][j+1] + 0.9*(img4[i][j+1]+img4[i+1][j])-0.81*img4[i][j
```

```
    img6[i+1][j+1] = 0.01*img5[i+1][j+1] + 0.9*(img6[i][j+1]+img6[i+1][j])-0.81*img6[i][j
}

/* set up structure for output achromatic image */
/* to allocate a full color image use type 'c' */
get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );

/* set up structure for output color image */
/* Note that the type is 'c' rather than 'g' */
get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

/* copy rgb component to new images */
for ( i = 0; i < input_img.height; i++ )
for ( j = 0; j < input_img.width; j++ ) {
  pixel = (int32_t)img2[i+1][j+1];
  if(pixel>255) {
    red_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) green_img.mono[i][j] = 0;
    else red_img.mono[i][j] = pixel;
  }

  pixel = (int32_t)img4[i+1][j+1];
  if(pixel>255) {
    green_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) green_img.mono[i][j] = 0;
    else green_img.mono[i][j] = pixel;
  }

  pixel = (int32_t)img6[i+1][j+1];
  if(pixel>255) {
    blue_img.mono[i][j] = 255;
  }
  else {
    if(pixel<0) blue_img.mono[i][j] = 0;
    else blue_img.mono[i][j] = pixel;
  }
  color_img.color[0][i][j] = red_img.mono[i][j];
  color_img.color[1][i][j] = green_img.mono[i][j];
  color_img.color[2][i][j] = blue_img.mono[i][j];
}


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

```
  /* write color image */
  if ( write_TIFF ( fp, &color_img ) ) {
      fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
      exit ( 1 );
  }

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

  /* de-allocate space which was used for the images */
  free_TIFF ( &(input_img) );
  free_TIFF ( &(green_img) );
  free_TIFF ( &(red_img) );
  free_TIFF ( &(blue_img) );
  free_TIFF ( &(color_img) );

  free_img( (void**)img1 );
  free_img( (void**)img2 );
  free_img( (void**)img3 );
  free_img( (void**)img4 );
  free_img( (void**)img5 );
  free_img( (void**)img6 );

  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);
}
```

MATLAB code for section 3,4,5

```
clc;
close all;
clear all;

u = -pi:0.05:pi;
v = -pi:0.05:pi;
[U,V] = meshgrid(u,v);

%% Section 3
H1 = zeros(size(U));
for k = -4:4
    for l = -4:4
        H1 = H1 + 1/81*exp(-1i*(k.*U+l.*V));
    end
```

```
end
H1 = abs(H1);
figure;
surf(U,V,H1)
xlabel('U')
ylabel('V')
zlabel('H(U,V)')
title('DSFT of h(m,n)')
colormap
colorbar

%% Section 4
H2 = zeros(size(U));
for k = -2:2
    for l = -2:2
        H2 = H2 + 1/25*exp(-1i*(k.*U+l.*V));
    end
end
H2 = abs(H2);
figure;
surf(U,V,H2)
xlabel('U')
ylabel('V')
zlabel('H(U,V)')
title('DSFT of h(m,n)')
colormap
colorbar

lamada = 1.5;
G2 = abs((1+lamada)*ones(size(H2))-lamada*H2);
figure;
surf(U,V,G2)
xlabel('U')
ylabel('V')
zlabel('G(U,V)')
title('DSFT of g(m,n)')
colormap
colorbar

%% Section 5
H3 = abs(0.01./(1-0.9*exp(-i*U)-0.9*exp(-i*V)+0.81*exp(-i*U).*exp(-i*V)));
figure;
surf(U,V,H3)
xlabel('U')
ylabel('V')
zlabel('H(U,V)')
title('DSFT of h(m,n)-IIR')
colormap
colorbar


%% View and compare

OriginImg = imread('img03.tif');
```

```
BlurImg = imread('imgblur.tif');
image(OriginImg);
image(BlurImg);
colormap(gray(256));
truesize;

%% Compute the point spread function h(m,n)
x = zeros(256,256);
x(127,127) = 1;
y = zeros(256,256);
for k = 2:256
    for l = 2:256
        y(k,l) = 0.01*x(k,l) + 0.9*(y(k,l-1)+y(k-1,l))- 0.81*y(k-1,l-1);
    end
end
h = y;
imwrite(uint8(255*100*h),'h_out.tif')
```