

ECE 5273

HW 5 Solution

Spring 2021

Dr. Havlicek

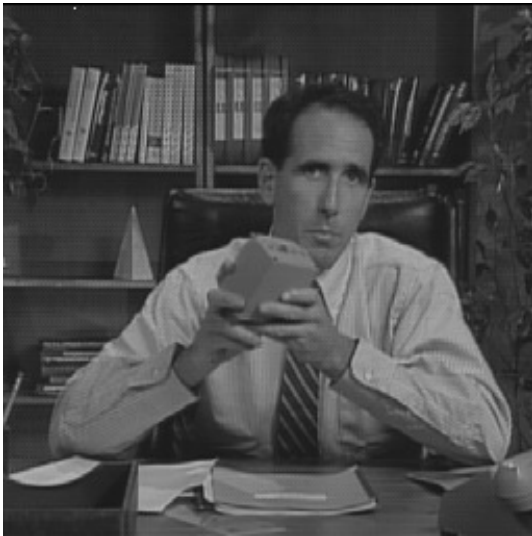
Note: This document contains solutions in both Matlab and traditional C.

Matlab Solution:

1.

(a)

Original Image

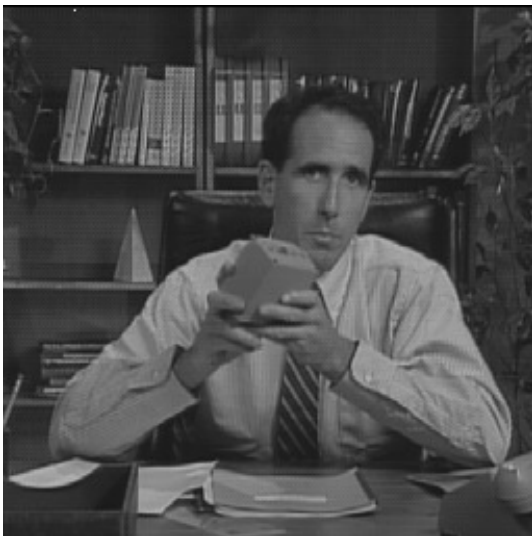


Filtered Image

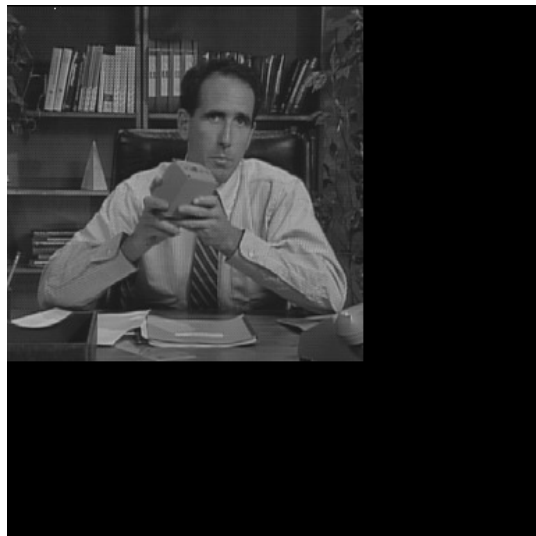


(b)

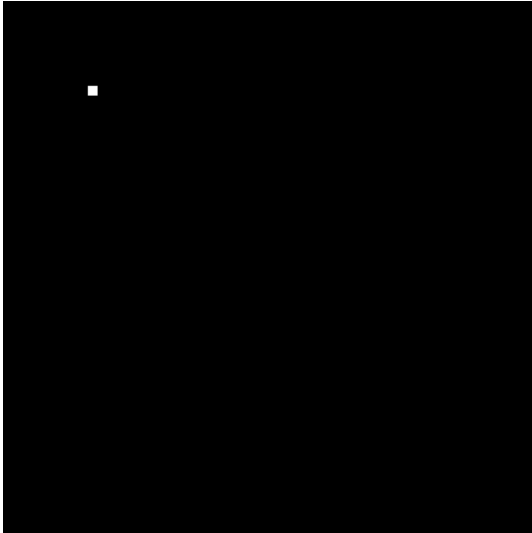
Original Image



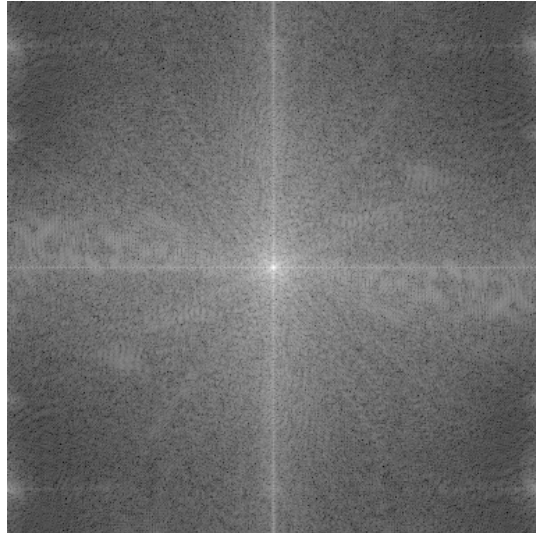
Zero Padded Image



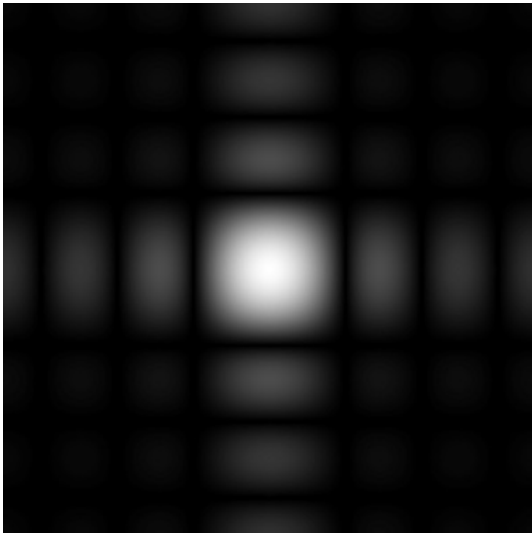
Zero Padded Impulse Resp



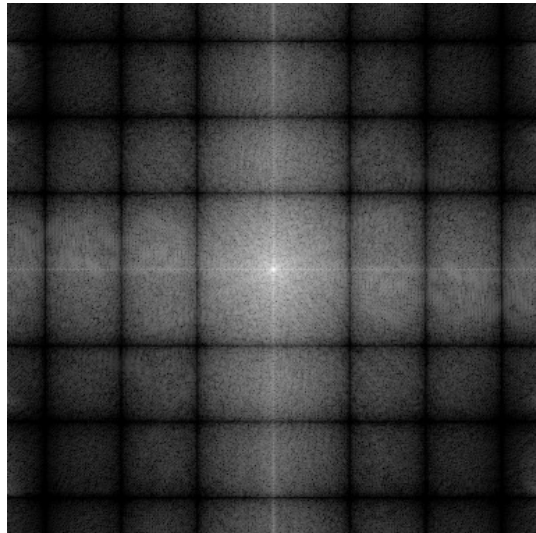
Log-magnitude spectrum of zero padded image



Log-magnitude spectrum of zero padded H image



Log-magnitude spectrum of zero padded result



Zero Padded Result



Final Filtered Image

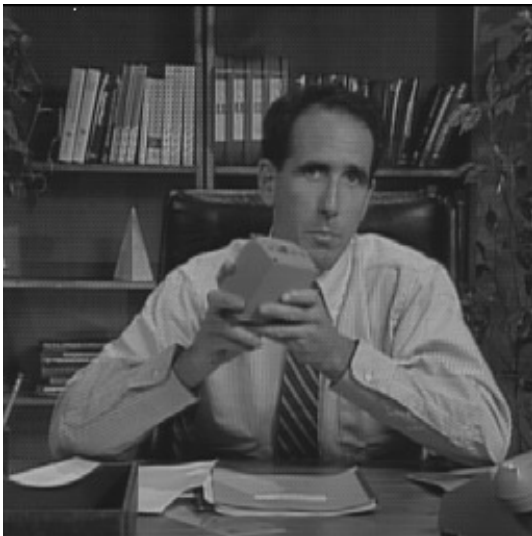


Matlab command window output:

```
(b): max difference from part (a): 0
```

(c)

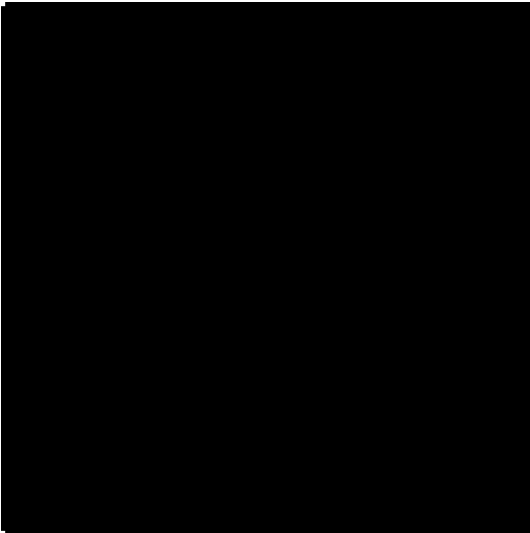
Original Image



Zero Phase Impulse Resp



Zero Padded zero-phase H



Final Filtered Image



Matlab command window output:

```
(c): max difference from part (a): 0
```

Matlab m-file listing:

```
%  
% P1.m  
%  
% 04/05/2015 jph  
%  
  
%-----  
%  
% P1(a): apply a 7x7 average filter to salesman image by doing linear  
% convolution in the image domain.  
%  
% Handle edge effects by zero padding.  
%  
  
X = ReadBin('salesman.bin',256);  
figure(1);image(X);colormap(gray(256));axis('image','off');  
title('Original Image','FontSize',18);  
print -deps Msalesman.eps;  
  
X2 = zeros(262,262);  
X2(4:259,4:259) = X;  
Y2 = zeros(262,262);  
  
for row=4:259  
    for col=4:259
```

```

        Y2(row,col) = sum(sum(X2(row-3:row+3,col-3:col+3)))/49;
    end
end

Y = stretch(Y2(4:259,4:259));
figure(2);image(Y);colormap(gray(256));axis('image','off');
title('Filtered Image','FontSize',18);
print -deps MY1a.eps;

% Save the result image for comparison with later results
Y1a = Y;

%-----
%
% P1(b): Use the method of Example 3 on page 5.61 of the Notes to do
% the same linear convolution by pointwise multiplication of DFT's.
%
% The impulse response image H will be 128x128.
%
% The original image is already in the Matlab array X.
%

% Make the 128x128 impulse response image
H = zeros(128,128);
H(62:68,62:68) = 1/49;

% Zero pad the original image and the H image
Padsz = 256 + 128 - 1;
ZPX = zeros(Padsz,Padsz);
ZPX(1:256,1:256) = X;
figure(3);image(ZPX);colormap(gray(256));axis('image','off');
title('Zero Padded Image','FontSize',18);
print -deps MZPX1b.eps;

ZPH = zeros(Padsz,Padsz);
ZPH(1:128,1:128) = H;
figure(4);image(stretch(ZPH));colormap(gray(256));axis('image','off');
title('Zero Padded Impulse Resp','FontSize',18);
print -deps MZPH1b.eps;

% Compute DFT's of zero padded images
ZPXtilde = fft2(ZPX);
ZPHtilde = fft2(ZPH);

% Show centered log-magnitude spectra
ZPXtildeDisplay = stretch(log(1 + abs(fftshift(ZPXtilde))));
figure(5);image(ZPXtildeDisplay);colormap(gray(256));axis('image','off');
title('Log-magnitude spectrum of zero padded image','FontSize',12);

```

```

print -deps MZPXtilde1b.eps;
ZPHtildeDisplay = stretch(log(1 + abs(fftshift(ZPHtilde))));
figure(6);image(ZPHtildeDisplay);colormap(gray(256));axis('image','off');
title('Log-magnitude spectrum of zero padded H image','FontSize',12);
print -deps MZPHtilde1b.eps;

% Compute the convolution by pointwise multiplication of DFT's.
% Show the resulting zero padded image and it's centered log-magnitude
% spectrum.
ZPYtilde = ZPXtilde .* ZPHtilde;
ZPY = ifft2(ZPYtilde);

ZPYtildeDisplay = stretch(log(1 + abs(fftshift(ZPYtilde))));
figure(7);image(ZPYtildeDisplay);colormap(gray(256));axis('image','off');
title('Log-magnitude spectrum of zero padded result','FontSize',12);
print -deps MZPYtilde1b.eps;
figure(8);image(stretch(ZPY));colormap(gray(256));axis('image','off');
title('Zero Padded Result','FontSize',18);
print -deps MZPY1b.eps;

% Extract the final result image and display
Y = stretch(ZPY(65:320,65:320));
figure(9);image(Y);colormap(gray(256));axis('image','off');
title('Final Filtered Image','FontSize',18);
print -deps MY1b.eps;

% Compare this result image with the one from part (a)
disp(['(b): max difference from part (a): ' num2str(max(max(abs(Y-Y1a))))])

%-----
%
% P1(c): Use the method of Example 5 on page 5.76 of the Notes to do
% the same linear convolution again by pointwise multiplication of
% DFT's, this time using a 256x256 true zero-phase impulse response.
%
% The impulse response image H will be 256x256.
%
% The original image is still in the Matlab array X.
%

% Make the 256x256 impulse response image H
% Put the "square" in the center
H = zeros(256,256);
H(126:132,126:132) = 1/49;

% Now use fftshift to get the true zero-phase impulse response image
H2 = fftshift(H);
figure(10);image(stretch(H2));colormap(gray(256));axis('image','off');

```

```

title('Zero Phase Impulse Resp','FontSize',18);
print -deps MH1c.eps;

% Zero pad the input image
ZPX = zeros(512,512);
ZPX(1:256,1:256) = X;

% Make the zero padded impulse response image as in Example 5
% on page 5.76 of the notes
ZPH2 = zeros(256,256);
ZPH2(1:128,1:128) = H2(1:128,1:128);
ZPH2(1:128,385:512) = H2(1:128,129:256);
ZPH2(385:512,1:128) = H2(129:256,1:128);
ZPH2(385:512,385:512) = H2(129:256,129:256);
figure(11);image(stretch(ZPH2));colormap(gray(256));axis('image','off');
title('Zero Padded zero-phase H','FontSize',18);
print -deps MZPH1c.eps;

% Compute the filtered result by pointwise multiplication of DFT's
Y = ifft2(fft2(ZPX) .* fft2(ZPH2));
Y = stretch(Y(1:256,1:256));
figure(12);image(Y);colormap(gray(256));axis('image','off');
title('Final Filtered Image','FontSize',18);
print -deps MY1c.eps;

% Compare this result image with the one from part (a)
disp(['(c): max difference from part (a): ' num2str(max(max(abs(Y-Y1a))))])

```

```

%
% ReadBin.m
%
% Read a square raw BYTE image (one byte per pixel, no header) from disk
% into a matlab array.
%
% Usage:
% >> x = ReadBin(fn,xsize);
%
% Input parameters:
%   fn          input filename
%   xsize       number of rows/cols in the image
%
% Output parameters:
%   x           double output array, holds the image
%
% 4/3/03 jph
%

function [x] = ReadBin(fn,xsize)

%
% Open the file
%
fid = fopen(fn,'r');
if (fid == -1)
    error(['Could not open ',fn]);
end;

%
% Read and close the file
%
[x,Nread] = fread(fid,[xsize,xsize],'uchar');
if (Nread ~= xsize*xsize)
    error(['Complete read of ',fn,' did not succeed.']);
end;
fclose(fid);

%
% Transpose data for matlab's 'row major' convention and return
%
x = x';

```



```

%
% stretch.m
%
% Perform a full-scale contrast stretch on a byte-per-pixel gray
% scale image.
%
% Usage:
% >> y = stretch(x);
%
% Input parameters:
%     x           double array, holds the input image
%
% Output parameters:
%     y           double array, holds the output image
%
% 4/3/03 jph
%

function [y] = stretch(x)

%
% Find the extremes and compute the scale factor
%
xMax = max(max(x));
xMin = min(min(x));
ScaleFactor = 255.0 / (xMax - xMin);

%
% Do the full-scale stretch
%
y = round((x - xMin) * ScaleFactor);

```

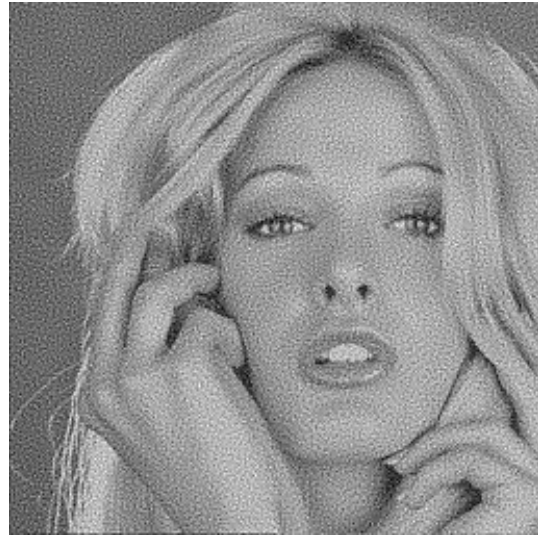
2.

(a)

Original Tiffany Image



girl2Noise32Hi



girl2Noise32



Matlab command window output:

```
MSE girl2Noise32Hi.bin: 692.505  
MSE girl2Noise32.bin: 744.4679
```

(b)

LPF on girl2



LPF on Noise32Hi



LPF on Noise32



Matlab command window output:

```
MSE: ideal LPF on girl2:      127.7481
MSE: ideal LPF on Noise32Hi: 398.9978
ISNR: ideal LPF on Noise32Hi: 2.3945 dB
MSE: ideal LPF on Noise32:   550.8787
ISNR: ideal LPF on Noise32:  1.3079 dB
```

(c)

Gauss1 on girl2



Gauss1 on Noise32Hi



Gauss1 on Noise32



Matlab command window output:

```
MSE: Gaussian LPF on girl2:      91.2585
MSE: Gaussian LPF on Noise32Hi: 415.8219
ISNR: Gaussian LPF on Noise32Hi: 2.2152 dB
MSE: Gaussian LPF on Noise32:   534.7741
ISNR: Gaussian LPF on Noise32:  1.4368 dB
```

(d)

Gauss2 on girl2



Gauss2 on Noise32Hi



Gauss2 on Noise32



Matlab command window output:

```
MSE: Gaussian2 LPF on girl2:      60.6184
MSE: Gaussian2 LPF on Noise32Hi: 406.7896
ISNR: Gaussian2 LPF on Noise32Hi: 2.3105 dB
MSE: Gaussian2 LPF on Noise32:   527.771
ISNR: Gaussian2 LPF on Noise32:   1.494 dB
```

Matlab m-file listing:

```
%
% P2.m
%
% 04/05/2015 jph
%

% original girl2 image (aka tiffany)
X = ReadBin('girl2.bin',256);

% with broadband noise
XN = ReadBin('girl2Noise32.bin',256);

% with hi pass noise
XNhi = ReadBin('girl2Noise32Hi.bin',256);

%-----
%
% P2(a): Display images and compute the MSE of each noisy image
%

xx = (XNhi - X).^2;
MSE_Nhi = mean(xx(:));
disp(['MSE girl2Noise32Hi.bin:   ' num2str(MSE_Nhi)])
xx = (XN - X).^2;
MSE_N = mean(xx(:));
disp(['MSE girl2Noise32.bin:     ' num2str(MSE_N)])
disp(' ');

figure(1);image(X);colormap(gray(256));axis('image','off');
title('Original Tiffany Image','FontSize',18);
print -deps Mgirl2.eps;
figure(2);image(XNhi);colormap(gray(256));axis('image','off');
title('girl2Noise32Hi','FontSize',18);
print -deps Mgirl2Noise32Hi.eps;
figure(3);image(XN);colormap(gray(256));axis('image','off');
title('girl2Noise32','FontSize',18);
print -deps Mgirl2Noise32.eps;

%-----
%
% P2(b): Apply isotropic ideal LPF with U_cutoff = 64.
% Use circular convolution for this.
%

U_cutoff = 64;
[U,V] = meshgrid(-128:127,-128:127);
```

```

HLtildeCenter = double(sqrt(U.^2 + V.^2) <= U_cutoff);
HLtilde = fftshift(HLtildeCenter);

% apply to original girl2 image and compute MSE
Y1 = ifft2(fft2(X) .* HLtilde);
yy = (Y1 - X).^2;
MSE_Y1 = mean(yy(:));
disp(['MSE: ideal LPF on girl2:      ' num2str(MSE_Y1)])

% apply to image with hi pass noise; compute MSE and ISNR
Y1Nhi = ifft2(fft2(XNhi) .* HLtilde);
yy = (Y1Nhi - X).^2;
MSE_Y1Nhi = mean(yy(:));
disp(['MSE: ideal LPF on Noise32Hi: ' num2str(MSE_Y1Nhi)])
ISNR_Y1Nhi = 10*log10(MSE_Nhi/MSE_Y1Nhi);
disp(['ISNR: ideal LPF on Noise32Hi: ' num2str(ISNR_Y1Nhi) ' dB'])

% apply to image with broadband noise; compute MSE and ISNR
Y1N = ifft2(fft2(XN) .* HLtilde);
yy = (Y1N - X).^2;
MSE_Y1N = mean(yy(:));
disp(['MSE: ideal LPF on Noise32:    ' num2str(MSE_Y1N)])
ISNR_Y1N = 10*log10(MSE_N/MSE_Y1N);
disp(['ISNR: ideal LPF on Noise32:   ' num2str(ISNR_Y1N) ' dB'])
disp(' ');

figure(4);image(stretch(Y1));colormap(gray(256));axis('image','off');
title('LPF on girl2','FontSize',18);
print -deps MY1.eps;
figure(5);image(stretch(Y1Nhi));colormap(gray(256));axis('image','off');
title('LPF on Noise32Hi','FontSize',18);
print -deps MY1Nhi.eps;
figure(6);image(stretch(Y1N));colormap(gray(256));axis('image','off');
title('LPF on Noise32','FontSize',18);
print -deps MY1N.eps;

%-----
%
% P2(c): Use the method of Example 4 p. 5.65 to apply Gaussian
% LPF with U_cutoff = 64.
%
U_cutoff_G = 64;
SigmaG = 0.19 * 256 / U_cutoff_G;
[U,V] = meshgrid(-128:127,-128:127);
GtildeCenter = exp((-2*pi^2*SigmaG^2)/(256.^2)*(U.^2 + V.^2));
Gtilde = fftshift(GtildeCenter);
G = ifft2(Gtilde);
G2 = fftshift(G);

```

```

ZPG2 = zeros(512,512);
ZPG2(1:256,1:256) = G2;

% apply to original girl2 image and compute MSE
ZPX = zeros(512,512);
ZPX(1:256,1:256) = X;
yy = ifft2(fft2(ZPX).*fft2(ZPG2));
Y2 = yy(129:384,129:384);
yy = (Y2 - X).^2;
MSE_Y2 = mean(yy(:));
disp(['MSE: Gaussian LPF on girl2:      ' num2str(MSE_Y2)])

% apply to image with hi pass noise; compute MSE and ISNR
ZPX = zeros(512,512);
ZPX(1:256,1:256) = XNhi;
yy = ifft2(fft2(ZPX).*fft2(ZPG2));
Y2Nhi = yy(129:384,129:384);
yy = (Y2Nhi - X).^2;
MSE_Y2Nhi = mean(yy(:));
disp(['MSE: Gaussian LPF on Noise32Hi: ' num2str(MSE_Y2Nhi)])
ISNR_Y2Nhi = 10*log10(MSE_Nhi/MSE_Y2Nhi);
disp(['ISNR: Gaussian LPF on Noise32Hi: ' num2str(ISNR_Y2Nhi) ' dB'])

% apply to image with broadband noise; compute MSE and ISNR
ZPX = zeros(512,512);
ZPX(1:256,1:256) = XN;
yy = ifft2(fft2(ZPX).*fft2(ZPG2));
Y2N = yy(129:384,129:384);
yy = (Y2N - X).^2;
MSE_Y2N = mean(yy(:));
disp(['MSE: Gaussian LPF on Noise32:   ' num2str(MSE_Y2N)])
ISNR_Y2N = 10*log10(MSE_N/MSE_Y2N);
disp(['ISNR: Gaussian LPF on Noise32:   ' num2str(ISNR_Y2N) ' dB'])
disp(' ');

figure(7);image(stretch(Y2));colormap(gray(256));axis('image','off');
title('Gauss1 on girl2','FontSize',18);
print -deps MY2.eps;
figure(8);image(stretch(Y2Nhi));colormap(gray(256));axis('image','off');
title('Gauss1 on Noise32Hi','FontSize',18);
print -deps MY2Nhi.eps;
figure(9);image(stretch(Y2N));colormap(gray(256));axis('image','off');
title('Gauss1 on Noise32','FontSize',18);
print -deps MY2N.eps;

%-----
%
% P2(d): Use the method of Example 4 p. 5.65 to apply Gaussian
% LPF with U_cutoff = 77.5.

```



```

%

U_cutoff_G = 77.5;
SigmaG = 0.19 * 256 / U_cutoff_G;
GtildeCenter = exp((-2*pi^2*SigmaG^2)/(256.^2)*(U.^2 + V.^2));
Gtilde = fftshift(GtildeCenter);
G = ifft2(Gtilde);
G2 = fftshift(G);
ZPG2 = zeros(512,512);
ZPG2(1:256,1:256) = G2;

% apply to original girl2 image and compute MSE
ZPX = zeros(512,512);
ZPX(1:256,1:256) = X;
yy = ifft2(fft2(ZPX).*fft2(ZPG2));
Y3 = yy(129:384,129:384);
yy = (Y3 - X).^2;
MSE_Y3 = mean(yy(:));
disp(['MSE: Gaussian2 LPF on girl2:      ' num2str(MSE_Y3)])

% apply to image with hi pass noise; compute MSE and ISNR
ZPX = zeros(512,512);
ZPX(1:256,1:256) = XNhi;
yy = ifft2(fft2(ZPX).*fft2(ZPG2));
Y3Nhi = yy(129:384,129:384);
yy = (Y3Nhi - X).^2;
MSE_Y3Nhi = mean(yy(:));
disp(['MSE: Gaussian2 LPF on Noise32Hi: ' num2str(MSE_Y3Nhi)])
ISNR_Y3Nhi = 10*log10(MSE_Nhi/MSE_Y3Nhi);
disp(['ISNR: Gaussian2 LPF on Noise32Hi: ' num2str(ISNR_Y3Nhi) ' dB'])

% apply to image with broadband noise; compute MSE and ISNR
ZPX = zeros(512,512);
ZPX(1:256,1:256) = XN;
yy = ifft2(fft2(ZPX).*fft2(ZPG2));
Y3N = yy(129:384,129:384);
yy = (Y3N - X).^2;
MSE_Y3N = mean(yy(:));
disp(['MSE: Gaussian2 LPF on Noise32:   ' num2str(MSE_Y3N)])
ISNR_Y3N = 10*log10(MSE_N/MSE_Y3N);
disp(['ISNR: Gaussian2 LPF on Noise32:   ' num2str(ISNR_Y3N) ' dB'])

figure(10);image(stretch(Y3));colormap(gray(256));axis('image','off');
title('Gauss2 on girl2','FontSize',18);
print -deps MY3.eps;
figure(11);image(stretch(Y3Nhi));colormap(gray(256));axis('image','off');
title('Gauss2 on Noise32Hi','FontSize',18);
print -deps MY3Nhi.eps;
figure(12);image(stretch(Y3N));colormap(gray(256));axis('image','off');

```

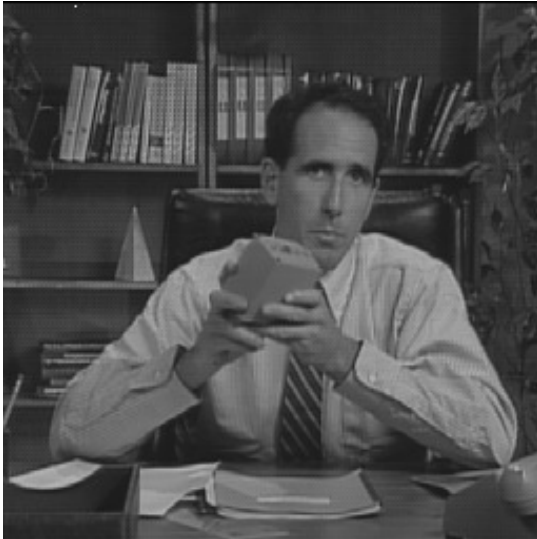
```
title('Gauss2 on Noise32','FontSize',18);  
print -deps MY3N.eps;
```

C Solution:

1.

(a)

Original Image

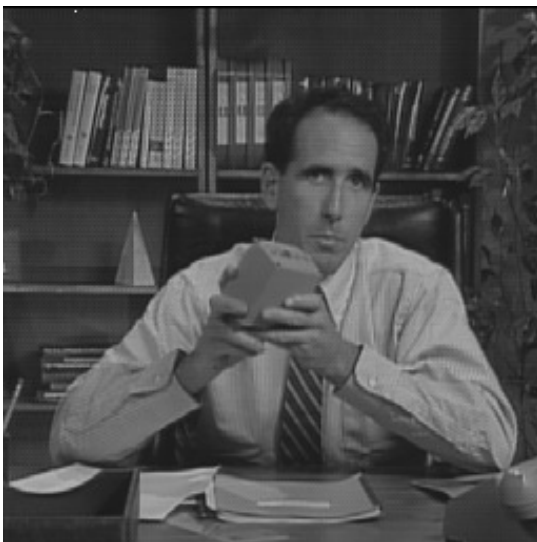


Filtered Image

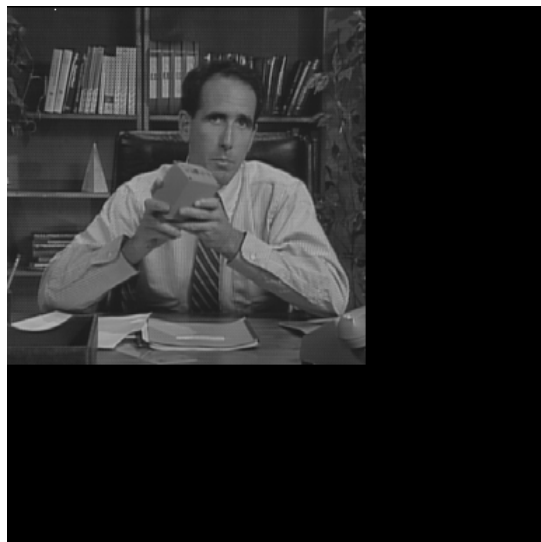


(b)

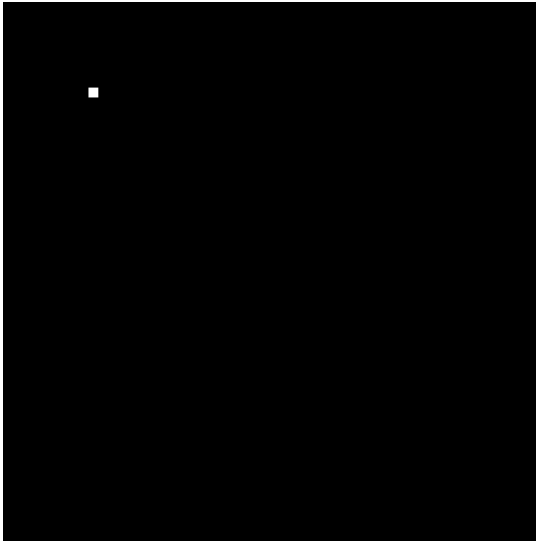
Original Image



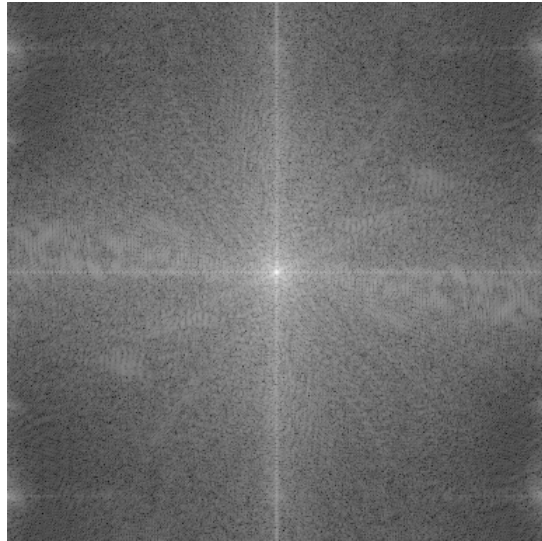
Zero Padded Image



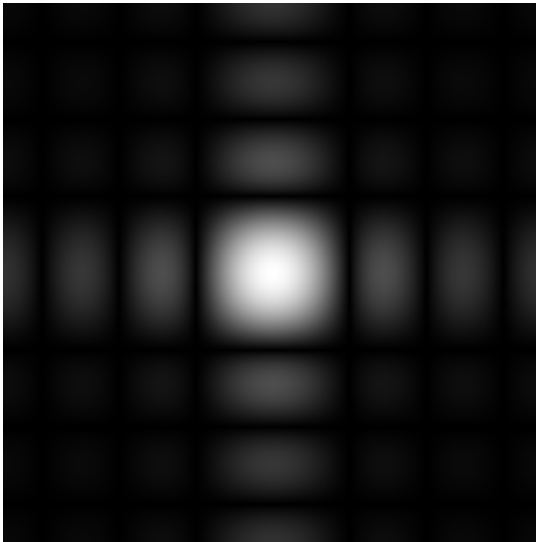
Zero Padded Impulse Response



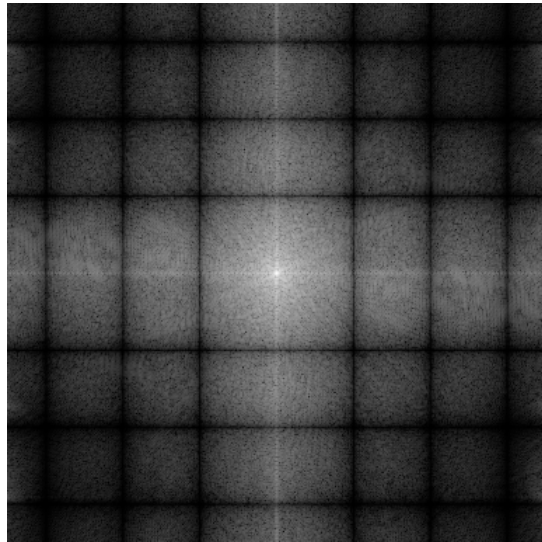
Log-Magnitude Spectrum of Zero Padded Image



Log-Magnitude Spectrum of Zero Padded H Image



Log-Magnitude Spectrum of Zero Padded Result



Zero Padded Result



Final Filtered Image

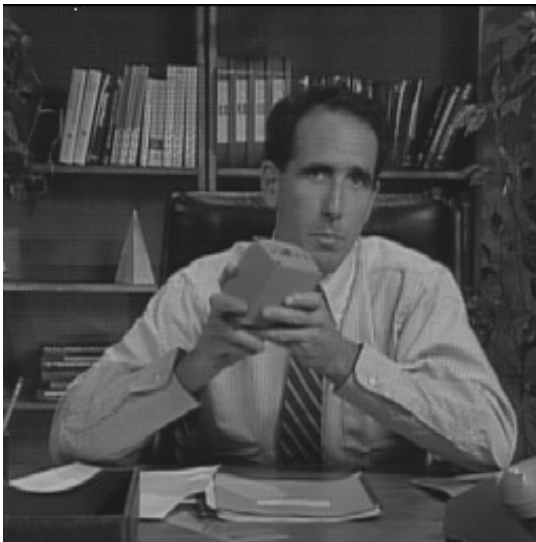


Console output:

(b): max difference from part (a): 0.00

(c)

Original Image



Zero Phase Impulse Response



Zero Padded Zero-Phase H Image



Final Filtered Image



Console output:

(c): max difference from part (a): 0.00

2.

(a)

Original Tiffany Image



girl2Noise32Hi



girl2Noise32



Console output:

MSE girl2Noise32Hi.bin: 692.5050

MSE girl2Noise32.bin: 744.4679

(b)

LPF on girl2



LPF on Noise32Hi



LPF on Noise32



Console output:

```
MSE: ideal LPF on girl2:      127.7481
MSE: ideal LPF on Noise32Hi: 398.9978
ISNR: ideal LPF on Noise32Hi: 2.3945 dB
MSE: ideal LPF on Noise32:   550.8787
ISNR: ideal LPF on Noise32:  1.3079 dB
```


(c)

Gauss1 on girl2



Gauss1 on Noise32Hi



Gauss1 on Noise32



Console output:

```
MSE: Gaussian LPF on girl2:      91.2585
MSE: Gaussian LPF on Noise32Hi: 415.8218
ISNR: Gaussian LPF on Noise32Hi: 2.2152 dB
MSE: Gaussian LPF on Noise32:   534.7740
ISNR: Gaussian LPF on Noise32:  1.4368 dB
```

(d)

Gauss2 on girl2



Gauss2 on Noise32Hi



Gauss2 on Noise32



Matlab command window output:

```
MSE: Gaussian2 LPF on girl2:      60.6184
MSE: Gaussian2 LPF on Noise32Hi: 406.7896
ISNR: Gaussian2 LPF on Noise32Hi: 2.3105 dB
MSE: Gaussian2 LPF on Noise32:   527.7709
ISNR: Gaussian2 LPF on Noise32:   1.4940 dB
```

C program listing:

```
//
// hw05.c
//
// This program requires an fftw3 installation. See the course web site for
// details of how to install.
//
// To compile:
// gcc hw05.c -o hw05 -lfftw3 -lm
//
// 3/22/2017 jph
//

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>

#include "fftw3.h"

#define BYTE unsigned char

//
// Function prototypes
//
void disk2byte();
void byte2disk();
void fft2d();
BYTE *f2b_FullScale();
float L2Norm();
void fCMult();
float fMSE();
float fISNR();
void PrintMaxDiffB();
BYTE *Problem1a();
BYTE *Problem1b();
BYTE *Problem1c();
void Problem2a();
void Problem2b();
void Problem2c();
void Problem2d();

/*-----*/
/* MAIN */
/*-----*/
```

```

main(argc,argv)

    int    argc;
    char   *argv[];
{

    BYTE   *y1a;                // BYTE result from 1(a)
    BYTE   *y1b;                // BYTE result from 1(b)
    BYTE   *y1c;                // BYTE result from 1(c)

    //
    // Problem 1
    //
    y1a = Problem1a(argv[0]);
    y1b = Problem1b(argv[0]);
    PrintMaxDiffB(y1a,y1b,256,'b','a');
    y1c = Problem1c(argv[0]);
    PrintMaxDiffB(y1a,y1c,256,'c','a');

    //
    // Problem 2
    //
    Problem2a(argv[0]);
    Problem2b(argv[0]);
    Problem2c(argv[0]);
    Problem2d(argv[0]);

    //
    // Collect the garbage
    //
    free(y1a);
    free(y1b);
    free(y1c);

    return;
} /*----- MAIN -----*/

/*-----
* Problem1a
*
*   HW 5, Problem 1(a).
*
*   - read in salesman.bin as a BYTE image
*   - apply a 7x7 average filter in the image domain
*     - handle edge effects by zero padding
*   - write output image as BYTE with full-scale contrast
*
*   jph 23 March 2017
*/

```

*

-----*/

BYTE *Problem1a(cmd)

```
char    *cmd;           // command used to invoke this program
{

int      i;             // loop counter
int      size;         // num rows/cols in image
int      size2;        // size + 6
int      N;            // size * size
int      N2;           // size2 * size2
int      row;          // row counter
int      row2;         // row counter in padded image
int      col;          // col counter
int      col2;         // col counter in padded image
int      Wrow;         // row in window
int      Wcol;         // col in window
float    one_on_49;    // 1/49
BYTE     *xb;          // BYTE input image
float    *xf2;         // zero padded float input image
BYTE     *yb;          // BYTE ouptput image
float    *yf;          // float output image; 256 x 256

printf("\nDoing Problem 1(a)...\n");
size = 256;
size2 = size + 6;
N = size * size;
N2 = size2 * size2;

//
// Allocate image arrays
//
if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(a).\n",cmd);
    exit(-1);
}
if ((xf2 = (float*)malloc(size2*size2*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(a).\n",cmd);
    exit(-1);
}
if ((yb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(a).\n",cmd);
    exit(-1);
}
if ((yf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(a).\n",cmd);
    exit(-1);
}
```

```

}

//
// Read the input image
//
disk2byte(xb,size,size,"salesman.bin");

//
// Cast to float and zero pad to 262 x 262
//
for (i=0; i < N2; i++) {
    xf2[i] = (float)0.0;
}
for (i=0,row=3; row < size2-3; row++) {
    for (col=3; col < size2-3; col++,i++) {
        xf2[row*size2 + col] = (float)xb[i];
    } // for row
} // for col

//
// Initialize output float image
//
for (i=0; i < N; i++) {
    yf[i] = (float)0.0;
}

//
// Apply the 7x7 average filter
//
one_on_49 = (float)1.0 / (float)49.0;
for (row=0,row2=3; row < size; row++,row2++) {
    for (col=0,col2=3; col < size; col++,col2++) {
        for (Wrow=row2-3; Wrow <= row2+3; Wrow++) {
            for (Wcol=col2-3; Wcol <= col2+3; Wcol++) {
                yf[row*size + col] += xf2[Wrow*size2 + Wcol];
            } // for Wcol
        } // for Wrow
        yf[row*size + col] = one_on_49 * yf[row*size + col];
    } // for col
} // for row

//
// Apply full-scale stretch to output image and cast to BYTE
//
yb = f2b_FullScale(yf,size,cmd);

//
// Write the output image to disk
//

```

```

byte2disk(yb,size,size,"CY1a.bin");

//
// Collect the garbage
//
free(xb);
free(xf2);
free(yf);

//
// Return the BYTE result
//
return(yb);
} /*----- Problem1a -----*/

/*-----
* Problem1b
*
* HW 5, Problem 1(b).
*
* Use the method of Example 3 on page 5.61 of the Notes to do the
* same linear convolution as in Problem 1(a) (7x7 average filter)
* by pointwise multiplication of DFT's.
*
* The impulse response image H will be 128x128.
*
* jph 23 March 2017
*
-----*/

BYTE *Problem1b(cmd)

char *cmd; // command used to invoke this program
{

int i; // loop counter
int size; // num rows/cols in input image
int Hsize; // num rows/cols in H image
int Padsiz; // size for zero padded images
int p0,q0; // horiz & vert offsets for H image
int row; // row counter
int col; // col counter
int row2; // row counter
int col2; // col counter
BYTE *xb; // BYTE input image
float *xf; // float input image; 256 x 256
float *ZPxR; // zero-padded input image, real part
float *ZPxI; // zero-padded input image, imag part
BYTE *BZPxR; // zero-padded input image; real; BYTE

```

```

float *XtildeR; // Real part of FFT of zero-padded image
float *XtildeI; // Imag part of FFT of zero-padded image
float *XtildeMag; // float log-magnitude spectrum of image
BYTE *BXtildeMag; // BYTE log-magnitude spectrum of image
float *H; // float impulse response image
float *ZPHR; // zero-padded H image, real part
float *ZPHI; // zero-padded H image, imag part
BYTE *BZPHR; // zero-padded H image; real; BYTE
float *HtildeR; // Real part of FFT of zero-padded H image
float *HtildeI; // Imag part of FFT of zero-padded H image
float *HtildeMag; // float log-magnitude spectrum of H image
BYTE *BHtildeMag; // BYTE log-magnitude spectrum of H image
float *YtildeR; // Real part of FFT of zero-padded output
float *YtildeI; // Imag part of FFT of zero-padded output
float *YtildeMag; // float log-magnitude spectrum of Y image
BYTE *BYtildeMag; // BYTE log-magnitude spectrum of Y image
float *ZPyR; // zero-padded output image, real part
float *ZPyI; // zero-padded output image, imag part
BYTE *BZPyR; // zero-padded output image; real; BYTE
float *yf; // float output image; 256 x 256
BYTE *Byf; // BYTE output image; 256 x 256

```

```
printf("\nDoing Problem 1(b)...\n");
```

```
size = 256;
```

```
Hsize = 128;
```

```
p0 = q0 = 64;
```

```
//
```

```
// Use a zero-padded size one larger than the minimum... see the
```

```
// note for problem 1(b) in the assignment for an explanation
```

```
//
```

```
Padsize = size + Hsize;
```

```
//
```

```
// Allocate image arrays
```

```
//
```

```
if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
```

```
if ((xf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
```

```
if ((ZPxR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
```

```
if ((ZPxI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
```



```

    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((XtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((XtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((XtildeMag = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((ZPHR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((ZPHI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((H = (float*)malloc(Hsize*Hsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((HtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((HtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((HtildeMag = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((YtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((YtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((YtildeMag = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
}

```

```

    exit(-1);
}
if ((ZPyR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((ZPyI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}
if ((yf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(b).\n",cmd);
    exit(-1);
}

//
// Read the input image
//
disk2byte(xb,size,size,"salesman.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
}

//
// Make the 128x128 impulse response image H
//
for (i=0; i < Hsize*Hsize; i++) {
    H[i] = (float)0.0;
}
for (row=q0-3; row <= q0+3; row++) {
    for (col=p0-3; col <= p0+3; col++) {
        H[row*Hsize + col] = (float)1.0/(float)49.0;
    }
}

//
// Zero pad both images
//
for (i=0; i < Padsize*Padsize; i++) {
    ZPxR[i] = (float)0.0;
    ZPxI[i] = (float)0.0;
    ZPHR[i] = (float)0.0;
    ZPHI[i] = (float)0.0;
}
for (row=0; row < size; row++) {

```

```

    for (col=0; col < size; col++) {
        ZPxR[row*Padsize + col] = xf[row*size + col];
    }
}
for (row=0; row < Hsize; row++) {
    for (col=0; col < Hsize; col++) {
        ZPHR[row*Padsize + col] = H[row*Hsize + col];
    }
}

//
// Write zero-padded images to disk as BYTE w/ full-scale contrast
//
BZPxR = f2b_FullScale(ZPxR,Padsize,cmd);
BZPHR = f2b_FullScale(ZPHR,Padsize,cmd);
byte2disk(BZPxR,Padsize,Padsize,"CZPX1b.bin");
byte2disk(BZPHR,Padsize,Padsize,"CZPH1b.bin");

//
// Compute DFT's of zero-padded images
//
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsize,1);
fft2d(ZPHR,ZPHI,HtildeR,HtildeI,Padsize,1);

//
// Write log-magnitude spectra of zero-padded input and H images
//
for (i=0; i < Padsize*Padsize; i++) {
    XtildeMag[i] = (float)log((double)1.0 + L2Norm(XtildeR[i],XtildeI[i]));
    HtildeMag[i] = (float)log((double)1.0 + L2Norm(HtildeR[i],HtildeI[i]));
}
BXtildeMag = f2b_FullScale(XtildeMag,Padsize,cmd);
BHtildeMag = f2b_FullScale(HtildeMag,Padsize,cmd);
byte2disk(BXtildeMag,Padsize,Padsize,"CZPXtilde1b.bin");
byte2disk(BHtildeMag,Padsize,Padsize,"CZPHtilde1b.bin");

//
// Compute convolution by pointwise multiplication of DFT's
//
for (i=0; i < Padsize*Padsize; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          HtildeR[i],HtildeI[i]);
}

//
// Write log-magnitude spectrum of zero-padded output image
//
for (i=0; i < Padsize*Padsize; i++) {
    YtildeMag[i] = (float)log((double)1.0 + L2Norm(YtildeR[i],YtildeI[i]));
}

```

```

}
BYtildeMag = f2b_FullScale(YtildeMag,Padsiz,cmd);
byte2disk(BYtildeMag,Padsiz,Padsiz,"CZPYtilde1b.bin");

//
// Compute zero-padded output image by inverse DFT and write to disk
//
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsiz,-1);
BZPyR = f2b_FullScale(ZPyR,Padsiz,cmd);
byte2disk(BZPyR,Padsiz,Padsiz,"CZPY1b.bin");

//
// Crop to get the final 256x256 output image and write to disk
//
for (row=0,row2=q0; row < size; row++,row2++) {
    for (col=0,col2=p0; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsiz + col2];
    }
}
Byf = f2b_FullScale(yf,size,cmd);
byte2disk(Byf,size,size,"CY1b.bin");

//
// Collect the garbage
//
free(H);
free(xb);
free(xf);
free(yf);
free(ZPxR);
free(ZPxI);
free(ZPHR);
free(ZPHI);
free(ZPyR);
free(ZPyI);
free(BZPxR);
free(BZPHR);
free(BZPyR);
free(XtildeR);
free(XtildeI);
free(HtildeR);
free(HtildeI);
free(YtildeR);
free(YtildeI);
free(XtildeMag);
free(HtildeMag);
free(YtildeMag);
free(BXtildeMag);
free(BHtildeMag);

```

```

free(BYtildeMag);

//
// Return the BYTE result
//
return(Byf);
} /*----- Problem1b -----*/

/*-----
* Problem1c
*
* HW 5, Problem 1(c).
*
* Use the method of Example 5 on page 5.76 of the Notes to do the
* same linear convolution as in Problems 1(a) and (b) (7x7 average
* filter) by pointwise multiplication of the DFT of the zero-padded image
* with the DFT of the correctly zero-padded zero-phase impulse response.
*
* The zero-phase impulse response image H will be 256x256.
*
* jph 25 March 2017
*
-----*/

```

BYTE *Problem1c(cmd)

```

char    *cmd;                // command used to invoke this program
{

int      i;                  // loop counter
int      size;              // num rows/cols in input image
int      so2;               // size / 2
int      Padsize;          // size for zero padded images
int      p0,q0;             // horiz & vert offsets for H image
int      row;               // row counter
int      col;               // col counter
BYTE     *xb;               // BYTE input image; 256 x 256
float     *xf;               // float input image; 256 x 256
float     *ZPxR;            // zero-padded input image, real part
float     *ZPxI;            // zero-padded input image, imag part
float     *XtildeR;         // Real part of FFT of zero-padded image
float     *XtildeI;         // Imag part of FFT of zero-padded image
float     *H;               // float impulse response image
float     *H2;              // float zero-phase impulse response image
float     *ZPH2R;           // zero-padded H2 image, real part
float     *ZPH2I;           // zero-padded H2 image, imag part
BYTE     *H2b;              // BYTE zero-phase impulse response image
BYTE     *ZPH2b;           // BYTE zero-padded H2 image
float     *H2tildeR;        // Real part of FFT of zero-padded H2 image

```

```

float   *H2tildeI;           // Imag part of FFT of zero-padded H2 image
float   *YtildeR;           // Real part of FFT of zero-padded output
float   *YtildeI;           // Imag part of FFT of zero-padded output
float   *ZPyR;              // zero-padded output image, real part
float   *ZPyI;              // zero-padded output image, imag part
float   *yf;                // float output image; 256 x 256
BYTE    *yb;                // BYTE output image; 256 x 256

```

```

printf("\nDoing Problem 1(c)...\n");

```

```

size = 256;
so2 = size/2;
Padsiz = size*2;
p0 = q0 = 128;

```

```

//
// Allocate image arrays
//
if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((xf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((ZPxR = (float*)malloc(Padsiz*Padsiz*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((ZPxI = (float*)malloc(Padsiz*Padsiz*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((XtildeR = (float*)malloc(Padsiz*Padsiz*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((XtildeI = (float*)malloc(Padsiz*Padsiz*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((H = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((H2 = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}

```

```

}
if ((ZPH2R = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((ZPH2I = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((H2tildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((H2tildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((YtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((YtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((ZPyR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((ZPyI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}
if ((yf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1(c).\n",cmd);
    exit(-1);
}

//
// Read the input image
//
disk2byte(xb,size,size,"salesman.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
}

```

```

//
// Make the 256x256 impulse response image H w/ the square in the center
//
for (i=0; i < size*size; i++) {
    H[i] = (float)0.0;
}
for (row=q0-3; row <= q0+3; row++) {
    for (col=p0-3; col <= p0+3; col++) {
        H[row*size + col] = (float)1.0/(float)49.0;
    }
}

//
// Permute the quadrants of the H image to get the true zero-phase
// impulse response image H2 as explained on pp. 5.68-5.71 of course Notes.
// This is equivalent to the Matlab fftshift routine.
//
for (row=0; row < so2; row++) {
    for (col=0; col < so2; col++) {

        // Quadrant I
        H2[row*size + col] = H[(row+so2)*size + (col+so2)];

        // Quadrant II
        H2[row*size + (col+so2)] = H[(row+so2)*size + col];

        // Quadrant III
        H2[(row+so2)*size + (col+so2)] = H[row*size + col];

        // Quadrant IV
        H2[(row+so2)*size + col] = H[row*size + (col+so2)];

    } // for col
} // for row

//
// Write H2 to disk as BYTE w/ full-scale contrast
//
H2b = f2b_FullScale(H2,size,cmd);
byte2disk(H2b,size,size,"CH1c.bin");

//
// Zero pad the input image
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPxR[i] = (float)0.0;
    ZPxI[i] = (float)0.0;
}

```



```

for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsiz + col] = xf[row*size + col];
    }
}

//
// Make the zero-padded impulse response image as in Example 5
// on page 5.76 of the notes.
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPH2R[i] = ZPH2I[i] = (float)0.0;
}
for (row=0; row < so2; row++) {
    for (col=0; col < so2; col++) {
        ZPH2R[row*Padsiz + col] = H2[row*size + col];
        ZPH2R[row*Padsiz + (Padsiz-so2+col)] = H2[row*size + (col+so2)];
        ZPH2R[(Padsiz-so2+row)*Padsiz + col] = H2[(row+so2)*size + col];
        ZPH2R[(Padsiz-so2+row)*Padsiz + (Padsiz-so2+col)]
            = H2[(row+so2)*size + (col+so2)];
    } // for col
} // for row

//
// Write zero-padded H2 image to disk w/ full-scale contrast
//
ZPH2b = f2b_FullScale(ZPH2R,Padsiz,cmd);
byte2disk(ZPH2b,Padsiz,Padsiz,"CZPH1c.bin");

//
// Compute the filtered image by pointwise multiplication of DFT's
//
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsiz,1);
fft2d(ZPH2R,ZPH2I,H2tildeR,H2tildeI,Padsiz,1);
for (i=0; i < Padsiz*Padsiz; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          H2tildeR[i],H2tildeI[i]);
}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsiz,-1);

//
// Crop to get the final 256x256 output image and write to disk
//
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        yf[row*size + col] = ZPyR[row*Padsiz + col];
    }
}
yb = f2b_FullScale(yf,size,cmd);

```

```

byte2disk(yb,size,size,"CY1c.bin");

//
// Collect the garbage
//
free(H);
free(H2);
free(xb);
free(xf);
free(yf);
free(H2b);
free(ZPxR);
free(ZPxI);
free(ZPyR);
free(ZPyI);
free(ZPH2R);
free(ZPH2I);
free(ZPH2b);
free(XtildeR);
free(XtildeI);
free(YtildeR);
free(YtildeI);
free(H2tildeR);
free(H2tildeI);

//
// Return the BYTE result
//
return(yb);
} /*----- Problem1c -----*/

/*-----
* Problem2a
*
* HW 5, Problem 2(a)
*
* Read images:
*   - girl2.bin
*   - girl2Noise32Hi.bin
*   - girl2Noise32.bin
*
* Compute MSE of girl2Noise32Hi.bin and girl2Noise32.bin
*
* jph 25 March 2017
*
-----*/

void Problem2a(cmd)

```

```

char    *cmd;                // command used to invoke this program
{

int      i;                  // loop counter
int      size;              // num rows/cols in input image
int      so2;               // size / 2
BYTE     *xb;               // BYTE image: girl2.bin
float    *xf;               // float image: girl2.bin
BYTE     *xNb;              // BYTE image: girl2Noise32.bin
float    *xNf;              // float image: girl2Noise32.bin
BYTE     *xNhib;            // BYTE image: girl2Noise32Hi.bin
float    *xNhif;            // float image: girl2Noise32Hi.bin
float    MSE_N;             // MSE of girl2Noise32.bin
float    MSE_Nhi;           // MSE of girl2Noise32Hi.bin

printf("\nDoing Problem 2(a)...\n");
size = 256;
so2 = size/2;

//
// Allocate image arrays
//
if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(a).\n",cmd);
    exit(-1);
}
if ((xf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(a).\n",cmd);
    exit(-1);
}
if ((xNb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(a).\n",cmd);
    exit(-1);
}
if ((xNf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(a).\n",cmd);
    exit(-1);
}
if ((xNhib = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(a).\n",cmd);
    exit(-1);
}
if ((xNhif = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(a).\n",cmd);
    exit(-1);
}

//
// Read images

```

```

//
disk2byte(xb,size,size,"girl2.bin");
disk2byte(xNb,size,size,"girl2Noise32.bin");
disk2byte(xNhib,size,size,"girl2Noise32Hi.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
    xNf[i] = (float)xNb[i];
    xNhif[i] = (float)xNhib[i];
}

//
// Compute MSE and print to console
//
MSE_Nhi = fMSE(xf,xNhif,size);
MSE_N = fMSE(xf,xNf,size);
printf("\nMSE girl2Noise32Hi.bin:  %.4f",MSE_Nhi);
printf("\nMSE girl2Noise32.bin:    %.4f\n",MSE_N);

//
// Collect the garbage
//
free(xb);
free(xf);
free(xNb);
free(xNf);
free(xNhib);
free(xNhif);

return;
} /*----- Problem2a -----*/

/*-----
* Problem2b
*
* HW 5, Problem 2(b)
*
* Read images:
*   - girl2.bin
*   - girl2Noise32Hi.bin
*   - girl2Noise32.bin
*
* Use circular convolution to apply an ideal isotropic LPF with
* U_cutoff = 64 cpi.
*
* jph 25 March 2017
*/

```

```

*
-----*/

void Problem2b(cmd)

    char    *cmd;                // command used to invoke this program
{

    int     i;                   // loop counter
    int     size;                // num rows/cols in input image
    int     so2;                 // size/2
    int     row;                 // row counter
    int     col;                 // col counter
    float   MSE;                 // mean squared error
    float   ISNR;                // improvement in SNR
    float   U_cutoff;            // LPF cutoff frequency in cpi
    float   U;                   // horiz freq in cpi
    float   V;                   // vert  freq in cpi
    BYTE    *xb;                 // BYTE  image: girl2.bin
    float   *xf;                 // float image: girl2.bin
    BYTE    *xNb;                // BYTE  image: girl2Noise32.bin
    float   *xNf;                // float image: girl2Noise32.bin
    BYTE    *xNhif;              // BYTE  image: girl2Noise32Hi.bin
    float   *xNhif;              // float image: girl2Noise32Hi.bin
    float   *HtildeCenter;       // centered frequency response image
    float   *fzero;              // floating point zero image
    float   *fzero2;             // floating point zero image
    float   *XtildeR;            // Real part of DFT of input image
    float   *XtildeI;            // Imag part of DFT of input image
    float   *YtildeR;            // Real part of DFT of output image
    float   *YtildeI;            // Imag part of DFT of output image
    float   *yf;                 // float output image
    BYTE    *yb;                 // BYTE  output image

    printf("\nDoing Problem 2(b)...\n");
    size = 256;
    so2 = size/2;

    //
    // Allocate image arrays
    //
    if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
        printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
        exit(-1);
    }
    if ((xf = (float*)malloc(size*size*sizeof(float))) == NULL) {
        printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
        exit(-1);
    }
}

```

```

if ((xNb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((xNf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((xNhib = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((xNhif = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((HtildeCenter = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((fzero = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((fzero2 = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((XtildeR = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((XtildeI = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((YtildeR = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((YtildeI = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
if ((yf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(b).\n",cmd);
    exit(-1);
}
}

```

```

//
// Read images
//
disk2byte(xb,size,size,"girl2.bin");
disk2byte(xNb,size,size,"girl2Noise32.bin");
disk2byte(xNhib,size,size,"girl2Noise32Hi.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
    xNf[i] = (float)xNb[i];
    xNhif[i] = (float)xNhib[i];
}
//
// Read images
//
disk2byte(xb,size,size,"girl2.bin");
disk2byte(xNb,size,size,"girl2Noise32.bin");
disk2byte(xNhib,size,size,"girl2Noise32Hi.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
    xNf[i] = (float)xNb[i];
    xNhif[i] = (float)xNhib[i];
}

//
// Design the centered frequency response image
//
U_cutoff = 64.0;
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        U = (float)(col - so2);
        V = (float)(row - so2);
        if (L2Norm(U,V) <= U_cutoff) {
            HtildeCenter[row*size + col] = (float)1.0;
        } else {
            HtildeCenter[row*size + col] = (float)0.0;
        } // else
    } // for col
} // for row

//
// Make floating point zero images

```

```

//
for (i=0; i < size*size; i++) {
    fzero[i] = fzero2[i] = (float)0.0;
}

//
// Use DFT to apply the filter to girl2
//
fft2d(xf,fzero,XtildeR,XtildeI,size,1);
for (i=0; i < size*size; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
           HtildeCenter[i],fzero[i]);
}
fft2d(yf,fzero2,YtildeR,YtildeI,size,-1);

//
// Compute MSE and write the output image to disk as BYTE
//
MSE = fMSE(xf,yf,size);
printf("\nMSE: ideal LPF on girl2:      %.4f",MSE);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY1.bin");
free(yb);

//
// Use DFT to apply the filter to girl2Noise32Hi
//
fft2d(xNhif,fzero,XtildeR,XtildeI,size,1);
for (i=0; i < size*size; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
           HtildeCenter[i],fzero[i]);
}
fft2d(yf,fzero2,YtildeR,YtildeI,size,-1);

//
// Compute MSE and ISNR; write output image as BYTE
//
MSE = fMSE(xf,yf,size);
printf("\nMSE: ideal LPF on Noise32Hi:  %.4f",MSE);
ISNR = fISNR(xf,xNhif,yf,size);
printf("\nISNR: ideal LPF on Noise32Hi:  %.4f dB",ISNR);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY1Nhi.bin");
free(yb);

//
// Use DFT to apply the filter to girl2Noise32
//
fft2d(xNf,fzero,XtildeR,XtildeI,size,1);

```



```

for (i=0; i < size*size; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          HtildeCenter[i],fzero[i]);
}
fft2d(yf,fzero2,YtildeR,YtildeI,size,-1);

//
// Compute MSE and ISNR; write output image as BYTE
//
MSE = fMSE(xf,yf,size);
printf("\nMSE: ideal LPF on Noise32:    %.4f",MSE);
ISNR = fISNR(xf,xNf,yf,size);
printf("\nISNR: ideal LPF on Noise32:    %.4f dB\n",ISNR);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY1N.bin");
free(yb);

//
// Collect the garbage
//
free(xb);
free(xf);
free(xNb);
free(xNf);
free(xNhib);
free(xNhif);
free(fzero);
free(fzero2);
free(XtildeR);
free(XtildeI);
free(YtildeR);
free(YtildeI);
free(HtildeCenter);

return;
} /*----- Problem2b -----*/

/*-----
* Problem2c
*
* HW 5, Problem 2(c)
*
* Read images:
*   - girl2.bin
*   - girl2Noise32Hi.bin
*   - girl2Noise32.bin
*
* Use the method of Example 4 on page 5.65 of the course notes
* to apply a Gaussian LPF with U_cutoff = 64 cpi.
*/

```

```

*
*   jph 25 March 2017
*

```

```

-----*/

```

```

void Problem2c(cmd)

```

```

    char    *cmd;                // command used to invoke this program
{

    int      i;                  // loop counter
    int      size;              // num rows/cols in input image
    int      so2;               // size/2
    int      Padsiz;           // size for zero padded images
    int      row;               // row counter
    int      col;               // col counter
    int      row2;              // row counter
    int      col2;              // col counter
    float    MSE;               // mean squared error
    float    ISNR;              // improvement in SNR
    float    U_cutoff_G;       // LPF cutoff frequency in cpi
    float    SigmaG;           // Gaussian filter space constant
    float    U;                 // horiz freq in cpi
    float    V;                 // vert  freq in cpi
    BYTE     *xb;               // BYTE  image: girl2.bin
    float    *xf;               // float image: girl2.bin
    BYTE     *xNb;              // BYTE  image: girl2Noise32.bin
    float    *xNf;              // float image: girl2Noise32.bin
    BYTE     *xNh;              // BYTE  image: girl2Noise32Hi.bin
    float    *xNh;              // float image: girl2Noise32Hi.bin
    float    *GtildeC;         // centered frequency response image
    float    *Gf;               // zero-phase impulse response image
    float    *G2f;             // centered impulse response image
    float    *ZPG2R;           // zero-padded G2 image, real part
    float    *ZPG2I;           // zero-padded G2 image, imag part
    float    *GtildeR;         // FFT of ZPG2, real part
    float    *GtildeI;         // FFT of ZPG2, imag part
    float    *ZPxR;            // zero padded input image, real part
    float    *ZPxI;            // zero padded input image, imag part
    float    *XtildeR;         // FFT of ZPx, imag part
    float    *XtildeI;         // FFT of ZPx, imag part
    float    *YtildeR;         // FFT of filtered image, real part
    float    *YtildeI;         // FFT of filtered image, imag part
    float    *ZPyR;            // zero-padded filtered image, real
    float    *ZPyI;            // zero-padded filtered image, imag
    float    *yf;               // float output image
    BYTE     *yb;               // BYTE  output image
    float    *fzero;           // floating point zero image
    float    *fzero2;          // floating point zero image

```

```

printf("\nDoing Problem 2(c)...\n");
size = 256;
so2 = size/2;
Padsize = size * 2;

//
// Allocate image arrays
//
if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((xf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((xNb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((xNf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((xNhib = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((xNhif = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((GtildeCenter = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((Gf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((G2f = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((ZPG2R = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
}

```

```

if ((ZPG2I = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((GtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((GtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((XtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((XtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((YtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((YtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((ZPxR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((ZPxI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((ZPyR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((ZPyI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((yf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((yb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {

```

```

    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((fzero = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}
if ((fzero2 = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(c).\n",cmd);
    exit(-1);
}

//
// Read images
//
disk2byte(xb,size,size,"girl2.bin");
disk2byte(xNb,size,size,"girl2Noise32.bin");
disk2byte(xNhib,size,size,"girl2Noise32Hi.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
    xNf[i] = (float)xNb[i];
    xNhif[i] = (float)xNhib[i];
}

//
// Make floating point zero images
//
for (i=0; i < size*size; i++) {
    fzero[i] = fzero2[i] = (float)0.0;
}

//
// Design the Gaussian LPF centered frequency response
//
U_cutoff_G = 64.0;
SigmaG = 0.19 * 256.0 / U_cutoff_G;
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        U = (float)(col - so2);
        V = (float)(row - so2);
        GtildeCenter[row*size + col] = (float)exp(
            (-2.0 * M_PI * M_PI * pow((double)SigmaG,(double)2.0))
            / pow((double)256.0,(double)2.0)
            * (pow((double)U,(double)2.0)

```

```

        + pow((double)V,(double)2.0));
    }    // for col
}    // for row

//
// Invert to obtain zero-phase impulse response image
//
fft2d(Gf,fzero2,GtildeCenter,fzero,size,-1);

//
// Permute the quadrants of the zero-phase impulse response
// image to center it
//
for (row=0; row < so2; row++) {
    for (col=0; col < so2; col++) {

        // Quadrant I
        G2f[row*size + col] = Gf[(row+so2)*size + (col+so2)];

        // Quadrant II
        G2f[row*size + (col+so2)] = Gf[(row+so2)*size + col];

        // Quadrant III
        G2f[(row+so2)*size + (col+so2)] = Gf[row*size + col];

        // Quadrant IV
        G2f[(row+so2)*size + col] = Gf[row*size + (col+so2)];

    }    // for col
}    // for row

//
// Zero pad the centered impulse response image
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPG2R[i] = ZPG2I[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPG2R[row*Padsiz + col] = G2f[row*size + col];
    }
}

//
// Compute the DFT of the zero-padded, centered impulse response
//
fft2d(ZPG2R,ZPG2I,GtildeR,GtildeI,Padsiz,1);

//

```

```

// Zero pad girl2, apply filter, write to disk, compute MSE
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPxR[i] = ZPxI[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsiz + col] = xf[row*size + col];
    }
}
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsiz,1);

for (i=0; i < Padsiz*Padsiz; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          GtildeR[i],GtildeI[i]);
}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsiz,-1);
for (row=0,row2=so2; row < size; row++,row2++) {
    for (col=0,col2=so2; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsiz + col2];
    }
}
MSE = fMSE(xf,yf,size);
printf("\nMSE: Gaussian LPF on girl2:      %.4f",MSE);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY2.bin");
free(yb);

//
// Zero pad girl2Noise32Hi, apply filter, write to disk, compute MSE & ISNR
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPxR[i] = ZPxI[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsiz + col] = xNhif[row*size + col];
    }
}
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsiz,1);

for (i=0; i < Padsiz*Padsiz; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          GtildeR[i],GtildeI[i]);
}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsiz,-1);
for (row=0,row2=so2; row < size; row++,row2++) {
    for (col=0,col2=so2; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsiz + col2];
    }
}

```

```

    }
}
MSE = fMSE(xf,yf,size);
printf("\nMSE: Gaussian LPF on Noise32Hi: %.4f",MSE);
ISNR = fISNR(xf,xNhif,yf,size);
printf("\nISNR: Gaussian LPF on Noise32Hi: %.4f dB",ISNR);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY2Nhi.bin");
free(yb);

//
// Zero pad girl2Noise32, apply filter, write to disk, compute MSE & ISNR
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPxR[i] = ZPxI[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsiz + col] = xNf[row*size + col];
    }
}
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsiz,1);

for (i=0; i < Padsiz*Padsiz; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          GtildeR[i],GtildeI[i]);
}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsiz,-1);
for (row=0,row2=so2; row < size; row++,row2++) {
    for (col=0,col2=so2; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsiz + col2];
    }
}
MSE = fMSE(xf,yf,size);
printf("\nMSE: Gaussian LPF on Noise32: %.4f",MSE);
ISNR = fISNR(xf,xNf,yf,size);
printf("\nISNR: Gaussian LPF on Noise32: %.4f dB\n",ISNR);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY2N.bin");
free(yb);

//
// Collect the garbage
//
free(xb);
free(xf);
free(yf);
free(Gf);
free(G2f);

```



```

    free(xNb);
    free(xNf);
    free(ZPxR);
    free(ZPxI);
    free(ZPyR);
    free(ZPyI);
    free(ZPG2R);
    free(ZPG2I);
    free(xNhib);
    free(xNhif);
    free(fzero);
    free(fzero2);
    free(GtildeR);
    free(GtildeI);
    free(XtildeR);
    free(XtildeI);
    free(YtildeR);
    free(YtildeI);

    return;
} /*----- Problem2c -----*/

/*-----
* Problem2d
*
*   HW 5, Problem 2(d)
*
*   Read images:
*   - girl2.bin
*   - girl2Noise32Hi.bin
*   - girl2Noise32.bin
*
*   Use the method of Example 4 on page 5.65 of the course notes
*   to apply a Gaussian LPF with U_cutoff = 77.5 cpi.
*
*   jph 25 March 2017
*
-----*/

void Problem2d(cmd)

    char    *cmd;                // command used to invoke this program
{

    int     i;                   // loop counter
    int     size;                // num rows/cols in input image
    int     so2;                 // size/2
    int     Padsizes;           // size for zero padded images
    int     row;                 // row counter

```

```

int    col;           // col counter
int    row2;         // row counter
int    col2;         // col counter
float  MSE;          // mean squared error
float  ISNR;         // improvement in SNR
float  U_cutoff_G;   // LPF cutoff frequency in cpi
float  SigmaG;       // Gaussian filter space constant
float  U;            // horiz freq in cpi
float  V;            // vert  freq in cpi
BYTE   *xb;          // BYTE  image: girl2.bin
float  *xf;          // float image: girl2.bin
BYTE   *xNb;         // BYTE  image: girl2Noise32.bin
float  *xNf;         // float image: girl2Noise32.bin
BYTE   *xNhib;       // BYTE  image: girl2Noise32Hi.bin
float  *xNhif;       // float image: girl2Noise32Hi.bin
float  *GtildeCenter; // centered frequency response image
float  *Gf;          // zero-phase impulse response image
float  *G2f;         // centered impulse response image
float  *ZPG2R;       // zero-padded G2 image, real part
float  *ZPG2I;       // zero-padded G2 image, imag part
float  *GtildeR;     // FFT of ZPG2, real part
float  *GtildeI;     // FFT of ZPG2, imag part
float  *ZPxR;        // zero padded input image, real part
float  *ZPxI;        // zero padded input image, imag part
float  *XtildeR;     // FFT of ZPx, imag part
float  *XtildeI;     // FFT of ZPx, imag part
float  *YtildeR;     // FFT of filtered image, real part
float  *YtildeI;     // FFT of filtered image, imag part
float  *ZPyR;        // zero-padded filtered image, real
float  *ZPyI;        // zero-padded filtered image, imag
float  *yf;          // float output image
BYTE   *yb;          // BYTE  output image
float  *fzero;       // floating point zero image
float  *fzero2;      // floating point zero image

printf("\nDoing Problem 2(d)...\n");
size = 256;
so2 = size/2;
Padsiz = size * 2;

//
// Allocate image arrays
//
if ((xb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((xf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);

```

```

    exit(-1);
}
if ((xNb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((xNf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((xNhib = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((xNhif = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((GtildeCenter = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((Gf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((G2f = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((ZPG2R = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((ZPG2I = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((GtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((GtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((XtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}

```

```

}
if ((XtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((YtildeR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((YtildeI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((ZPxR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((ZPxI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((ZPyR = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((ZPyI = (float*)malloc(Padsize*Padsize*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((yf = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((yb = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((fzero = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}
if ((fzero2 = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2(d).\n",cmd);
    exit(-1);
}

//
// Read images

```

```

//
disk2byte(xb,size,size,"girl2.bin");
disk2byte(xNb,size,size,"girl2Noise32.bin");
disk2byte(xNhib,size,size,"girl2Noise32Hi.bin");

//
// Cast to float
//
for (i=0; i < size*size; i++) {
    xf[i] = (float)xb[i];
    xNf[i] = (float)xNb[i];
    xNhif[i] = (float)xNhib[i];
}

//
// Make floating point zero images
//
for (i=0; i < size*size; i++) {
    fzero[i] = fzero2[i] = (float)0.0;
}

//
// Design the Gaussian LPF centered frequency response
//
U_cutoff_G = 77.5;
SigmaG = 0.19 * 256.0 / U_cutoff_G;
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        U = (float)(col - so2);
        V = (float)(row - so2);
        GtildeCenter[row*size + col] = (float)exp(
            (-2.0 * M_PI * M_PI * pow((double)SigmaG,(double)2.0))
            / pow((double)256.0,(double)2.0)
            * (pow((double)U,(double)2.0)
            + pow((double)V,(double)2.0)));
    } // for col
} // for row

//
// Invert to obtain zero-phase impulse response image
//
fft2d(Gf,fzero2,GtildeCenter,fzero,size,-1);

//
// Permute the quadrants of the zero-phase impulse response
// image to center it
//
for (row=0; row < so2; row++) {
    for (col=0; col < so2; col++) {

```

```

// Quadrant I
G2f[row*size + col] = Gf[(row+so2)*size + (col+so2)];

// Quadrant II
G2f[row*size + (col+so2)] = Gf[(row+so2)*size + col];

// Quadrant III
G2f[(row+so2)*size + (col+so2)] = Gf[row*size + col];

// Quadrant IV
G2f[(row+so2)*size + col] = Gf[row*size + (col+so2)];

} // for col
} // for row

//
// Zero pad the centered impulse response image
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPG2R[i] = ZPG2I[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPG2R[row*Padsiz + col] = G2f[row*size + col];
    }
}

//
// Compute the DFT of the zero-padded, centered impulse response
//
fft2d(ZPG2R,ZPG2I,GtildeR,GtildeI,Padsiz,1);

//
// Zero pad girl2, apply filter, write to disk, compute MSE
//
for (i=0; i < Padsiz*Padsiz; i++) {
    ZPxR[i] = ZPxI[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsiz + col] = xf[row*size + col];
    }
}
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsiz,1);

for (i=0; i < Padsiz*Padsiz; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          GtildeR[i],GtildeI[i]);
}

```

```

}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsize,-1);
for (row=0,row2=so2; row < size; row++,row2++) {
    for (col=0,col2=so2; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsize + col2];
    }
}
MSE = fMSE(xf,yf,size);
printf("\nMSE: Gaussian2 LPF on girl2:      %.4f",MSE);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY3.bin");
free(yb);

//
// Zero pad girl2Noise32Hi, apply filter, write to disk, compute MSE & ISNR
//
for (i=0; i < Padsize*Padsize; i++) {
    ZPxR[i] = ZPxI[i] = (float)0.0;
}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsize + col] = xNhif[row*size + col];
    }
}
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsize,1);

for (i=0; i < Padsize*Padsize; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          GtildeR[i],GtildeI[i]);
}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsize,-1);
for (row=0,row2=so2; row < size; row++,row2++) {
    for (col=0,col2=so2; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsize + col2];
    }
}
MSE = fMSE(xf,yf,size);
printf("\nMSE: Gaussian2 LPF on Noise32Hi: %.4f",MSE);
ISNR = fISNR(xf,xNhif,yf,size);
printf("\nISNR: Gaussian2 LPF on Noise32Hi: %.4f dB",ISNR);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY3Nhi.bin");
free(yb);

//
// Zero pad girl2Noise32, apply filter, write to disk, compute MSE & ISNR
//
for (i=0; i < Padsize*Padsize; i++) {
    ZPxR[i] = ZPxI[i] = (float)0.0;
}

```

```

}
for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
        ZPxR[row*Padsiz + col] = xNf[row*size + col];
    }
}
fft2d(ZPxR,ZPxI,XtildeR,XtildeI,Padsiz,1);

for (i=0; i < Padsiz*Padsiz; i++) {
    fCMult(&YtildeR[i],&YtildeI[i],XtildeR[i],XtildeI[i],
          GtildeR[i],GtildeI[i]);
}
fft2d(ZPyR,ZPyI,YtildeR,YtildeI,Padsiz,-1);
for (row=0,row2=so2; row < size; row++,row2++) {
    for (col=0,col2=so2; col < size; col++,col2++) {
        yf[row*size + col] = ZPyR[row2*Padsiz + col2];
    }
}
MSE = fMSE(xf,yf,size);
printf("\nMSE: Gaussian2 LPF on Noise32:   %.4f",MSE);
ISNR = fISNR(xf,xNf,yf,size);
printf("\nISNR: Gaussian2 LPF on Noise32:   %.4f dB\n",ISNR);
yb = f2b_FullScale(yf,size,cmd);
byte2disk(yb,size,size,"CY3N.bin");
free(yb);

//
// Collect the garbage
//
free(xb);
free(xf);
free(yf);
free(Gf);
free(G2f);
free(xNb);
free(xNf);
free(ZPxR);
free(ZPxI);
free(ZPyR);
free(ZPyI);
free(ZPG2R);
free(ZPG2I);
free(xNhib);
free(xNhif);
free(fzero);
free(fzero2);
free(GtildeR);
free(GtildeI);
free(XtildeR);

```



```

    free(XtildeI);
    free(YtildeR);
    free(YtildeI);

    return;
} /*----- Problem2d -----*/

/*-----
* disk2byte.c
*
*   function reads an unsigned char (byte) image from disk
*
*
*   jph 15 June 1992
*
-----*/

void disk2byte(x,row_dim,col_dim,fn)

    BYTE    *x;           /* image to be read */
    int     row_dim;     /* row dimension of x */
    int     col_dim;     /* col dimension of x */
    char    *fn;         /* filename */
{

    int fd;              /* file descriptor */
    int n_bytes;        /* number of bytes to read */

    /*
    * detect zero dimension input
    */
    if ((row_dim==0) || (col_dim==0)) return;

    /*
    * create and open the file
    */
    if ((fd = open(fn, O_RDONLY))== -1) {
        printf("\ndisk2byte.c : could not open %s !",fn);
        return;
    }

    /*
    * read image data from the file
    */
    n_bytes = row_dim * col_dim * sizeof(unsigned char);
    if (read(fd,x,n_bytes) != n_bytes) {
        printf("\ndisk2byte.c : complete read of %s did not succeed.",fn);
    }
}

```

```

/*
 * close file and return
 */
if (close(fd) == -1) printf("\ndisk2byte.c : error closing %s.",fn);
return;
} /*----- disk2byte -----*/

/*-----
 * byte2disk.c
 *
 * function writes an unsigned char (byte) image to disk
 *
 *
 * jph 15 June 1992
 *
-----*/

void byte2disk(x,row_dim,col_dim,fn)

    BYTE *x;           /* image to be written */
    int row_dim;       /* row dimension of x */
    int col_dim;       /* col dimension of x */
    char *fn;          /* filename */
{

    int fd;            /* file descriptor */
    int n_bytes;       /* number of bytes to read */

    /*
     * detect zero dimension input
     */
    if ((row_dim==0) || (col_dim==0)) return;

    /*
     * create and open the file
     */
    if ((fd = open(fn, O_WRONLY | O_CREAT | O_TRUNC, 0644))== -1) {
        printf("\nbyte2disk.c : could not open %s !",fn);
        return;
    }

    /*
     * write image data to the file
     */
    n_bytes = row_dim * col_dim * sizeof(unsigned char);
    if (write(fd,x,n_bytes) != n_bytes) {
        printf("\nbyte2disk.c : complete write of %s did not succeed.",fn);
    }
}

```

```

/*
 * close file and return
 */
if (close(fd) == -1) printf("\nbyte2disk.c : error closing %s.",fn);
return;
} /*----- byte2disk -----*/

/*-----
 * f2b_FullScale
 *
 * Convert a float image to a BYTE image with full-scale contrast.
 *
 *
 * jph 13 November 2000
 *
-----*/

BYTE *f2b_FullScale(x,size,cmd)

float   *x;                /* input float image */
int      size;             /* num rows/cols in input image x */
char     *cmd;             /* command used to invoke this program */
{

int      i;                /* loop counter */
int      N;                /* size * size */
float    min;              /* minimum pixel value in input image */
float    max;              /* maximum pixel value in input image */
float    ScaleFact;       /* dynamic range scale factor */
BYTE     *y;               /* output BYTE image */

N = size * size;

/*
 * Allocate output BYTE image on the heap
 */
if ((y = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\nFree store exhausted in f2b_FullScale.\n");
    exit(-1);
}

/*
 * find min and max pixel values in input image
 */
min = max = x[0];
for (i=0; i < N; i++) {
    if (x[i] < min) {
        min = x[i];

```

```

    } else {
        if (x[i] > max) {
            max = x[i];
        }
    }
}

/*
 * Do full-scale contrast stretch and cast image into BYTE
 */
ScaleFact = (float)255.0 / (max - min);
for (i=0; i < N; i++) {
    y[i] = (BYTE)round(ScaleFact * (x[i] - min));
}

return(y);
} /*----- f2b_FullScale -----*/

/*
 * fft2d.c
 *
 * Function implements the 2D FFT. FFTW3 is called to compute the FFT.
 * The frequency domain coordinates are as described in the FFT handout
 * given out in class. FFTW3 must be installed for this routine to work.
 *
 * The image must be square and the row/col dimension must be even.
 *
 * If you want to cut this routine and put it in a separate file so
 * that you can compile it separately from the main() and then link to
 * it, you need to #include "fftw3.h" and link -lfftw3 -lm.
 *
 * 2/28/07 jph
 */

void fft2d(xReal,xImag,XReal,XImag,size,direction)

float *xReal;      /* real part of signal */
float *xImag;      /* imaginary part of signal */
float *XReal;      /* real part of spectrum */
float *XImag;      /* imaginary part of spectrum */
int    size;       /* row/col dim of image */
int    direction; /* 1=fwd xform, -1=inverse xform */

{
    fftw_complex *pass;          /* array for passing data to/from fftw */
    fftw_plan Plan;             /* structure for fftw3 planning */
    float one_on_n;            /* 1 / (no. data points in signal) */
    int i;                      /* counter */

```

```

int n;                /* no. data points in signal */
int so2;             /* size over 2 */
int row,col;        /* loop counters */

n = size * size;
so2 = (size >> 1);
pass = fftw_malloc(n*sizeof(fftw_complex));

if (direction == 1 ) { /* forward transform */
    Plan = fftw_plan_dft_2d(size,size,pass,pass,FFTW_FORWARD,FFTW_ESTIMATE);
    /*
    * Set up the data in the pass array and call fftw3
    */
    for (i=0; i<n; i++) {
        pass[i][0] = xReal[i];
        pass[i][1] = xImag[i];
    }
    fftw_execute(Plan);

    /*
    * Center the spectrum returned by fftw3
    */
    for (row=0; row<so2; row++) {
        for (col=0; col<so2; col++) {

            // Quadrant I
            XReal[(row+so2)*size + col+so2] = pass[row*size + col][0];
            XImag[(row+so2)*size + col+so2] = pass[row*size + col][1];

            // Quadrant II
            XReal[(row+so2)*size + col] = pass[row*size + col+so2][0];
            XImag[(row+so2)*size + col] = pass[row*size + col+so2][1];

            // Quadrant III
            XReal[row*size + col] = pass[(row+so2)*size + col+so2][0];
            XImag[row*size + col] = pass[(row+so2)*size + col+so2][1];

            // Quadrant IV
            XReal[row*size + col+so2] = pass[(row+so2)*size + col][0];
            XImag[row*size + col+so2] = pass[(row+so2)*size + col][1];

        } // for col
    } // for row
} // if (direction == 1)

else {
    if (direction == -1) { /* reverse transform */
        Plan = fftw_plan_dft_2d(size,size,pass,pass,FFTW_BACKWARD,FFTW_ESTIMATE);
        one_on_n = (float)1.0 / (float)n;
    }
}

```

```

/*
 * "un" Center the given spectrum for passing to fftw3
 */
for (row=0; row<so2; row++) {
    for (col=0; col<so2; col++) {

        // Quadrant I
        pass[row*size + col][0] = XReal[(row+so2)*size + col+so2];
        pass[row*size + col][1] = XImag[(row+so2)*size + col+so2];

        // Quadrant II
        pass[row*size + col+so2][0] = XReal[(row+so2)*size + col];
        pass[row*size + col+so2][1] = XImag[(row+so2)*size + col];

        // Quadrant III
        pass[(row+so2)*size + col+so2][0] = XReal[row*size + col];
        pass[(row+so2)*size + col+so2][1] = XImag[row*size + col];

        // Quadrant IV
        pass[(row+so2)*size + col][0] = XReal[row*size + col+so2];
        pass[(row+so2)*size + col][1] = XImag[row*size + col+so2];

    } // for col
} // for row

fftw_execute(Plan);

/*
 * Copy data back out of pass array and scale
 */
for (i=0; i<n; i++) {
    xReal[i] = pass[i][0] * one_on_n;
    xImag[i] = pass[i][1] * one_on_n;
}
} // else

else {
    printf("\nERROR: fft2d: unknown value %d specified for direction.\n",
           direction);
    exit(-1);
}
}
fftw_destroy_plan(Plan);
fftw_free(pass);
return;
} /*----- fft2d -----*/

/*-----

```

```

* L2Norm
*
*   Return the L2 norm (Euclidean norm) of a floating point 2-vector.
*
*   jph 13 November 2000
*
-----*/

float L2Norm(x,y)

    float    x;          /* first element of input vector */
    float    y;          /* second element of input vector */
{

    return((float)pow(
                pow((double)x,(double)2.0) + pow((double)y,(double)2.0),
                (double)0.5));

} /*----- L2Norm -----*/

/*-----
* fCMult.c
*
*   Function computes a float complex product.  Output (yReal, yImag) is
*   set equal to the product of the inputs (aReal,aImag) * (bReal,bImag).
*
*   jph 9/13/93
-----*/

void fCMult(yReal,yImag,aReal,aImag,bReal,bImag)

    float *yReal;      /* pointer to real part of product */
    float *yImag;      /* pointer to imag part of product */
    float  aReal;      /* real part of multiplicand */
    float  aImag;      /* imag part of multiplicand */
    float  bReal;      /* real part of multiplier */
    float  bImag;      /* imag part of multiplier */

{
    *yReal = (aReal * bReal) - (aImag * bImag);
    *yImag = (aReal * bImag) + (aImag * bReal);
    return;
} /*----- fCMult -----*/

/*-----
* PrintMaxDiffB
*
*   Print out the max abs difference between two BYTE images.
*

```

```

*
*   jph 25 March 2017
*
-----*/

void PrintMaxDiffB(x1,x2,size,p1,p2)

    BYTE    *x1;           // BYTE image from part p1
    BYTE    *x2;           // BYTE image from part p2
    int     size;         // num rows/cols in input image x
    char    p1;           // letter designation of x1 part
    char    p2;           // letter designation of x2 part
{

    int     i;             // loop counter
    float   Dif;           // float difference btwn two pixels
    float   MaxDif;       // max abs difference between x1 and x2

    //
    // Loop over pixels; find the max abs diff between x1 and x2
    //
    MaxDif = (float)0.0;
    for (i=0; i < size*size; i++) {
        Dif = fabsf((float)x1[i] - (float)x2[i]);
        if (Dif > MaxDif) {
            MaxDif = Dif;
        }
    }

    //
    // Print result
    //
    printf("\n(%c): max difference from part (%c): %.2f\n",p1,p2,MaxDif);

    return;
} /*----- PrintMAXDiffB -----*/

/*-----
* fMSE
*
*   Compute MSE between two float images.
*
*   jph 25 March 2017
*
-----*/

float fMSE(x1,x2,size)

```



```

float  *x1;           // float input image
float  *x2;           // float input image
int    size;         // num rows/cols in input images
{

int    i;             // loop counter
double MSE;          // mean squared error

//
// Compute MSE
//
MSE = (double)0.0;
for (i=0; i < size*size; i++) {
    MSE += pow((double)(x1[i]-x2[i]),(double)2.0);
}
MSE = MSE / (double)(size*size);

return((float)MSE);
} /*----- fMSE -----*/

/*-----
* fISNR
*
*   Compute ISNR for a floating point filtering operation
*
*
*   jph 25 March 2017
*
*-----*/

float fISNR(x1,x2,y,size)

float  *x1;           // original image
float  *x2;           // corrupted image
float  *y;            // filtered image
int    size;         // num rows/cols in input images
{

return((float)((double)10.0 *
    log10((double)fMSE(x1,x2,size)/(double)fMSE(x1,y,size))));
} /*----- fISNR -----*/

```