ECE 5273
HW 4 Solution

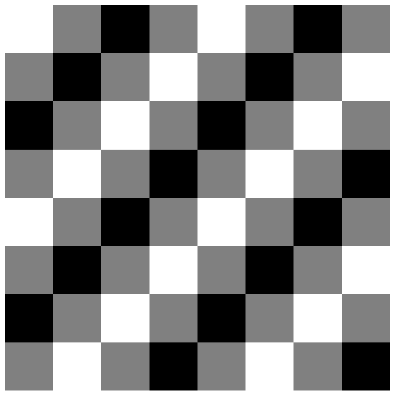
Spring 2025 Dr. Havlicek

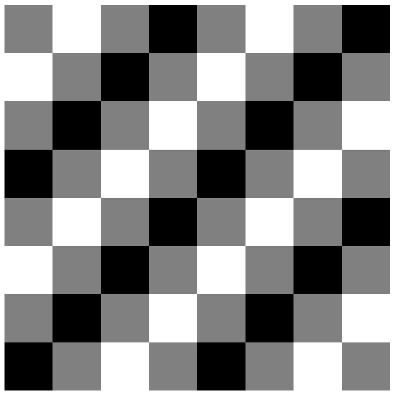
Note: This document contains solutions in both Matlab and traditional C.

C Solution:

1.

$\mathrm{Re}[\mathbf{I}_1]$ $\mathrm{Im}[\mathbf{I}_1]$





$\mathrm{Re}[\widetilde{\mathbf{I}}_1]$:

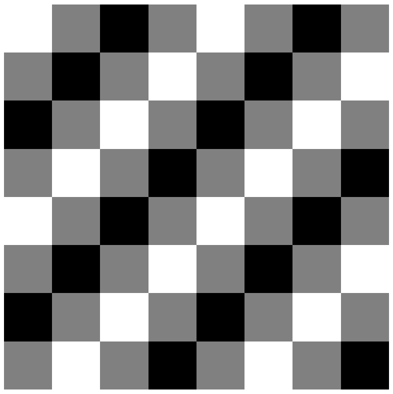
```
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000   -0.0000   -0.0000   -0.0000   32.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

$\mathrm{Im}[\widetilde{\mathbf{I}}_1]$:

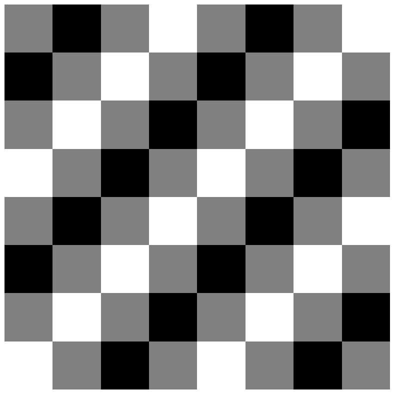
```
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

2.

$\mathrm{Re}[\mathbf{I}_2]$



$\mathrm{Im}[\mathbf{I}_2]$



$\mathrm{Re}[\widetilde{\mathbf{I}}_2]$:

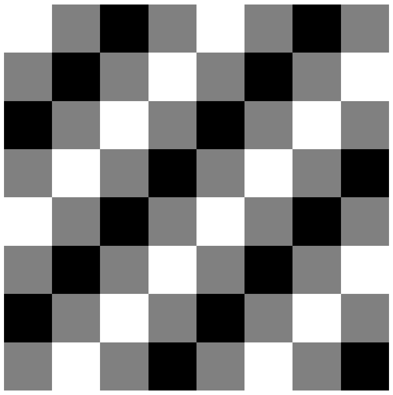
```
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   32.0000   -0.0000   -0.0000   -0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

$\mathrm{Im}[\widetilde{\mathbf{I}}_2]$:

```
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

3.

$\mathbf{I}_3$



Re[$\widetilde{\mathbf{I}}_3$]:

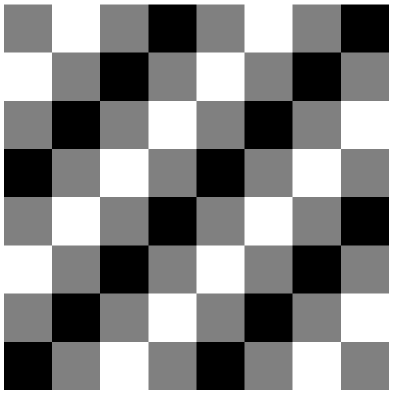
```
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   32.0000   -0.0000   -0.0000   -0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000   -0.0000   -0.0000   -0.0000   32.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

Im[$\widetilde{\mathbf{I}}_3$]:

```
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

4.

$\mathbf{I}_4$



$\mathrm{Re}[\widetilde{\mathbf{I}}_4]$:

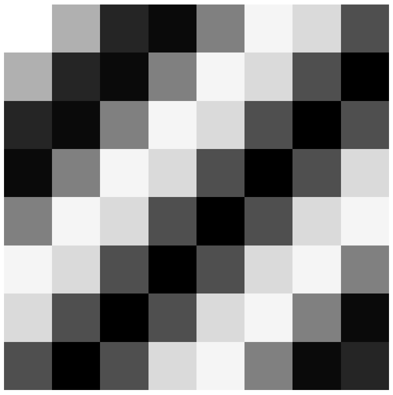
```
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

$\mathrm{Im}[\widetilde{\mathbf{I}}_4]$:

```
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
 0.0000    0.0000   32.0000   -0.0000   -0.0000   -0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
 0.0000    0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
-0.0000   -0.0000    0.0000    0.0000    0.0000    0.0000  -32.0000   -0.0000
 0.0000    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000    0.0000
```

5.

$\mathbf{I}_5$



$\mathrm{Re}[\widetilde{\mathbf{I}}_5]$:

```
 0.5535    0.4335   -0.6131    2.9028    2.0000    2.9028   -0.6131    0.4335
 0.4335   -0.1397   -2.7422    4.8284    2.2688    2.0000    0.6488    0.7023
-0.6131   -2.7422  -11.6569   13.5706    4.6131    2.1796    2.0000    0.6488
 2.9028    4.8284   13.5706  -11.8603   -3.2620   -2.3592    2.1796    2.0000
 2.0000    2.2688    4.6131   -3.2620   -1.2398   -3.2620    4.6131    2.2688
 2.9028    2.0000    2.1796   -2.3592   -3.2620  -11.8603   13.5706    4.8284
-0.6131    0.6488    2.0000    2.1796    4.6131   13.5706  -11.6569   -2.7422
 0.4335    0.7023    0.6488    2.0000    2.2688    4.8284   -2.7422   -0.1397
```

$\mathrm{Im}[\widetilde{\mathbf{I}}_5]$:

```
 0.0000    0.6488    2.6131   -2.1796    0.0000    2.1796   -2.6131   -0.6488
 0.6488    1.2977    3.2620   -1.5307    0.6488    2.8284   -1.9643   -0.0000
 2.6131    3.2620    5.2263    0.4335    2.6131    4.7927    0.0000    1.9643
-2.1796   -1.5307    0.4335   -4.3592   -2.1796   -0.0000   -4.7927   -2.8284
 0.0000    0.6488    2.6131   -2.1796    0.0000    2.1796   -2.6131   -0.6488
 2.1796    2.8284    4.7927    0.0000    2.1796    4.3592   -0.4335    1.5307
-2.6131   -1.9643    0.0000   -4.7927   -2.6131   -0.4335   -5.2263   -3.2620
-0.6488    0.0000    1.9643   -2.8284   -0.6488    1.5307   -3.2620   -1.2977
```

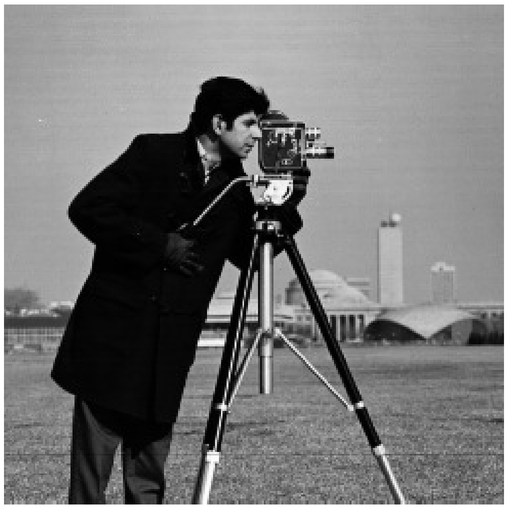
Discussion: For problems 1 and 2, the images $\mathbf{I}_1$ and $\mathbf{I}_2$ were each equal to one of the DFT basis functions $W_N^{-(um+vn)}$. Therefore, the DFTs $\widetilde{\mathbf{I}}_1$ and $\widetilde{\mathbf{I}}_2$ each had only one nonzero coefficient.

In problems 3 and 4, each image was a sum of two DFT basis functions. So each DFT had two nonzero coefficients.

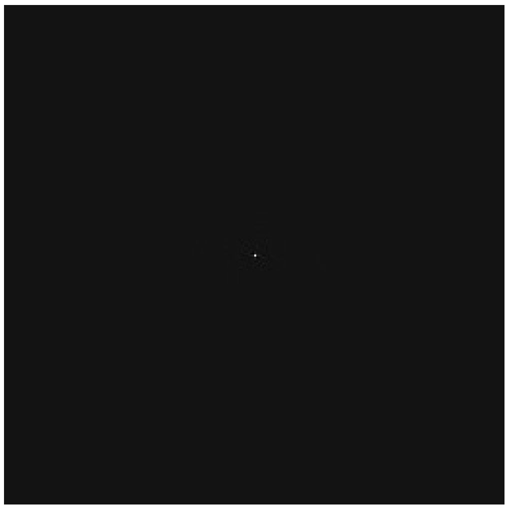
For problem 5, however, the frequency of $\mathbf{I}_5$ is distinct from that of any of the DFT basis functions. Therefore, many nonzero coefficients are required in the DFT to create the image $\mathbf{I}_5$ by constructive and destructive interference between all of the basis functions $W_N^{-(um+vn)}$.

6.

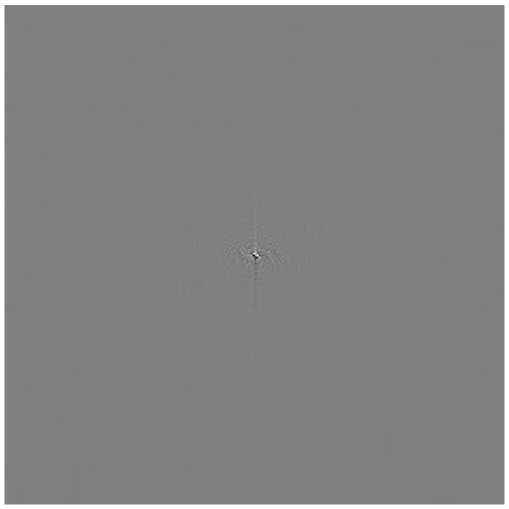
camera



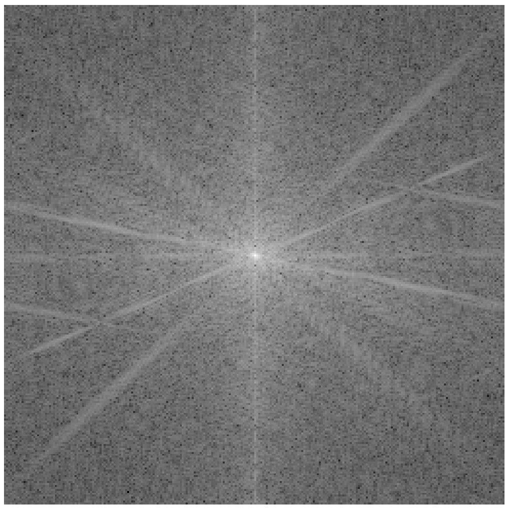
Re[DFT(camera)]



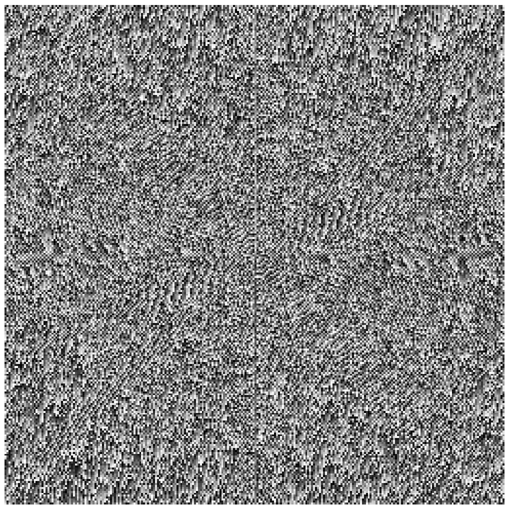
Im[DFT(camera)]



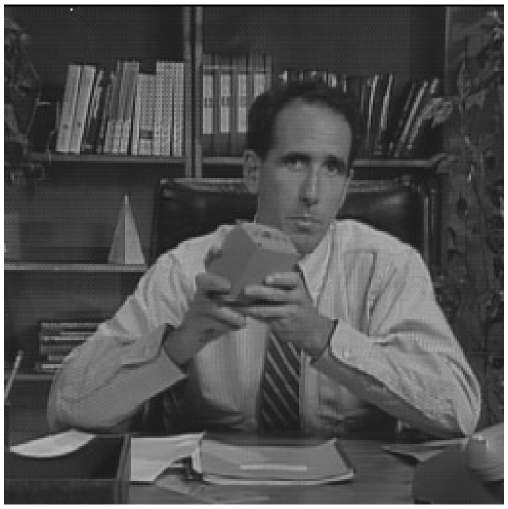
Log-Magnitude Spectrum



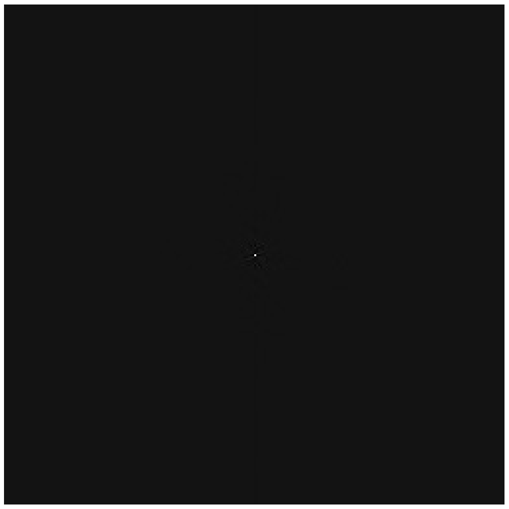
arg[DFT(camera)]



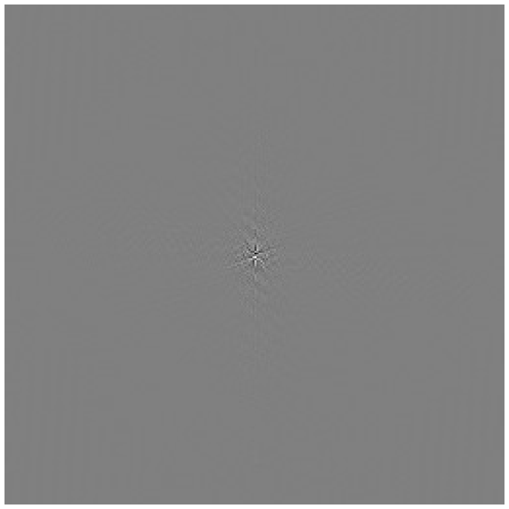
salesman



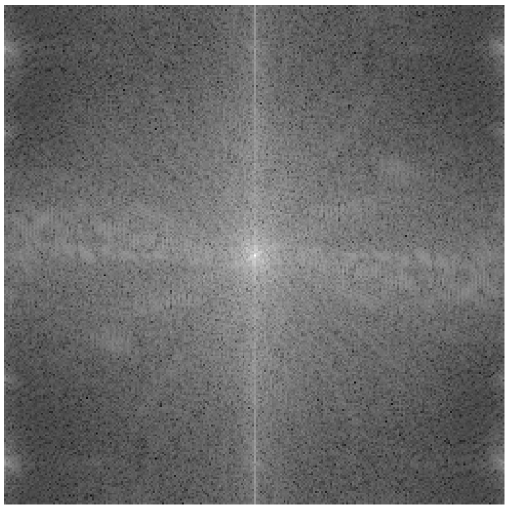
Re[DFT(salesman)]



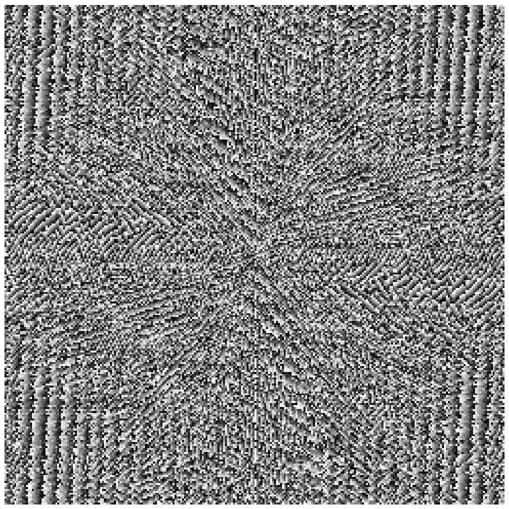
Im[DFT(salesman)]



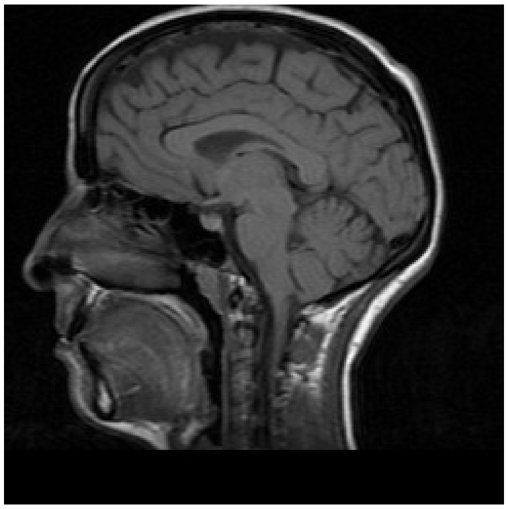
Log-Magnitude Spectrum



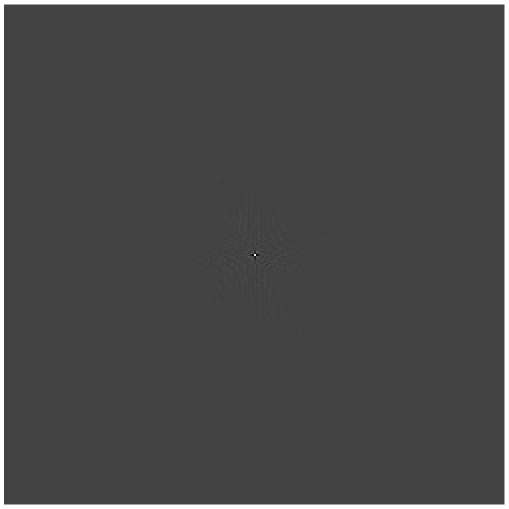
arg[DFT(salesman)]



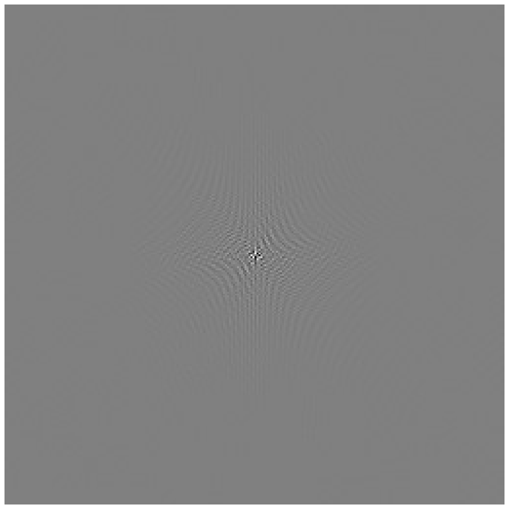
head



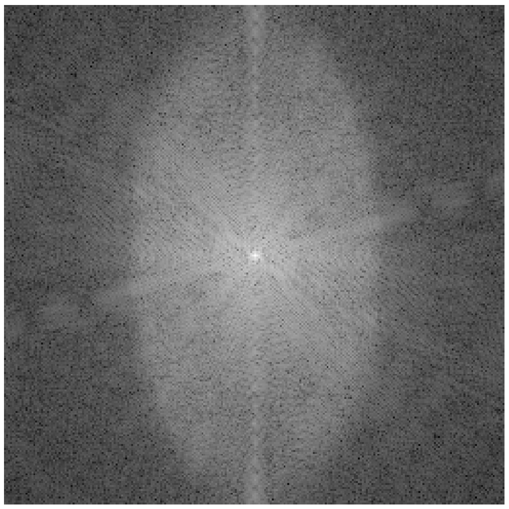
Re[DFT(head)]



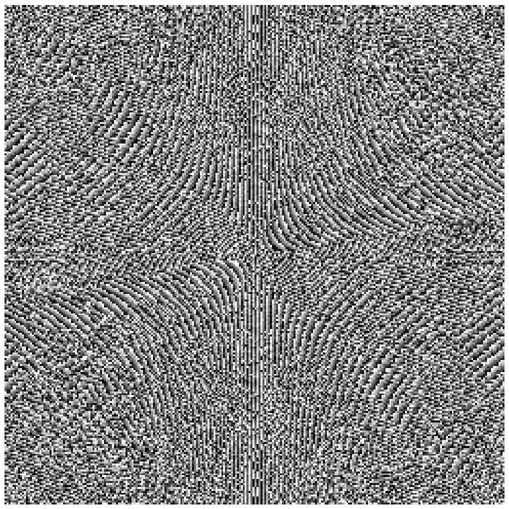
Im[DFT(head)]



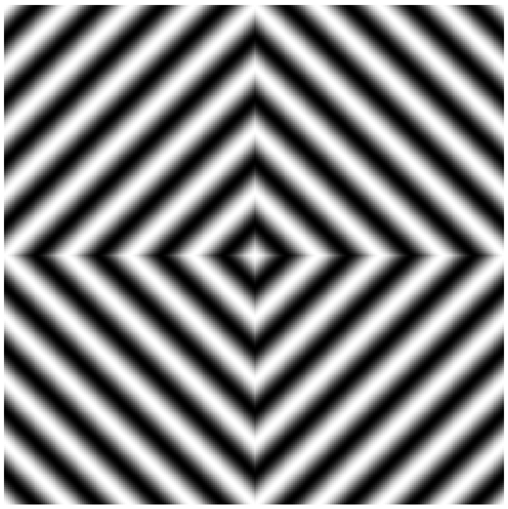
Log-Magnitude Spectrum



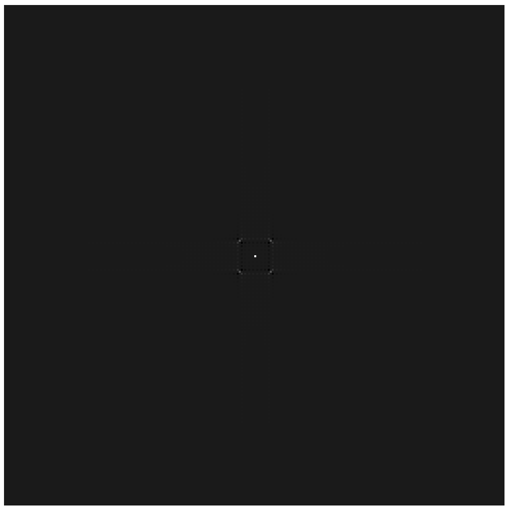
arg[DFT(head)]



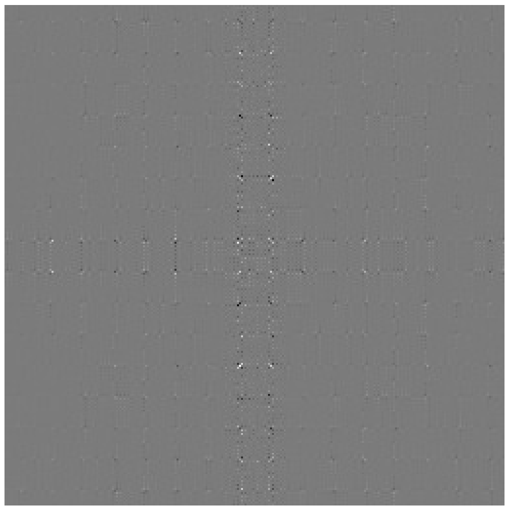
eyeR



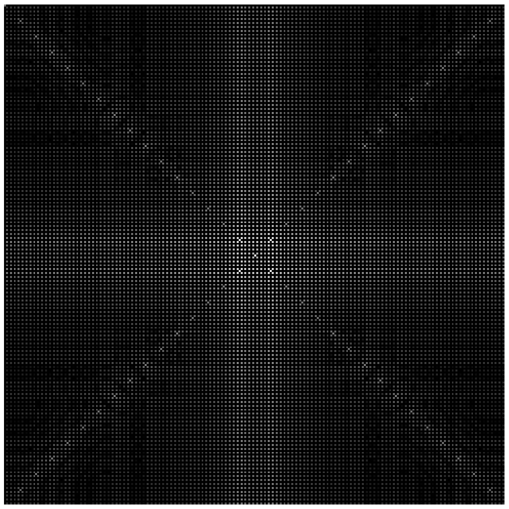
Re[DFT(eyeR)]



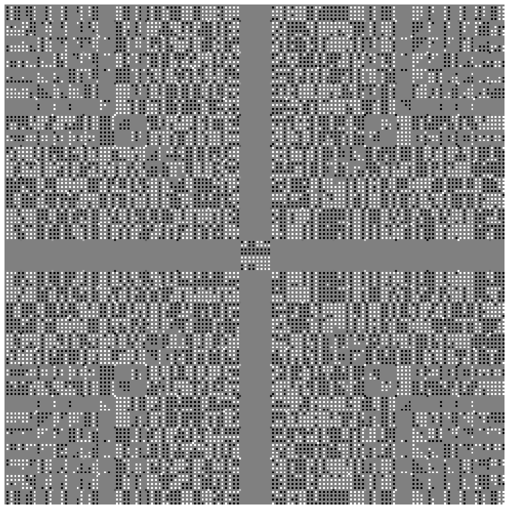
Im[DFT(eyeR)]



Log-Magnitude Spectrum

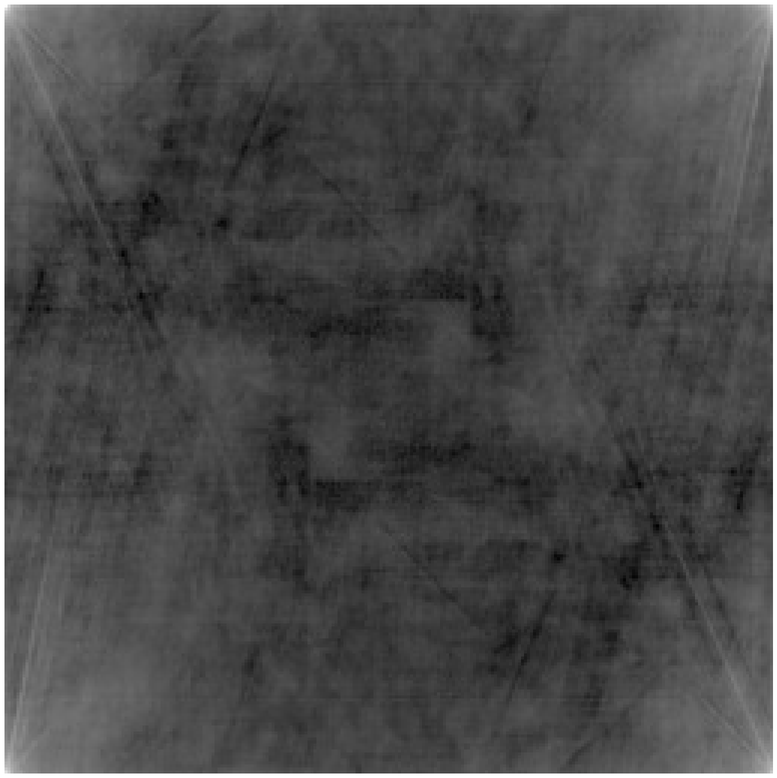


arg[DFT(eyeR)]

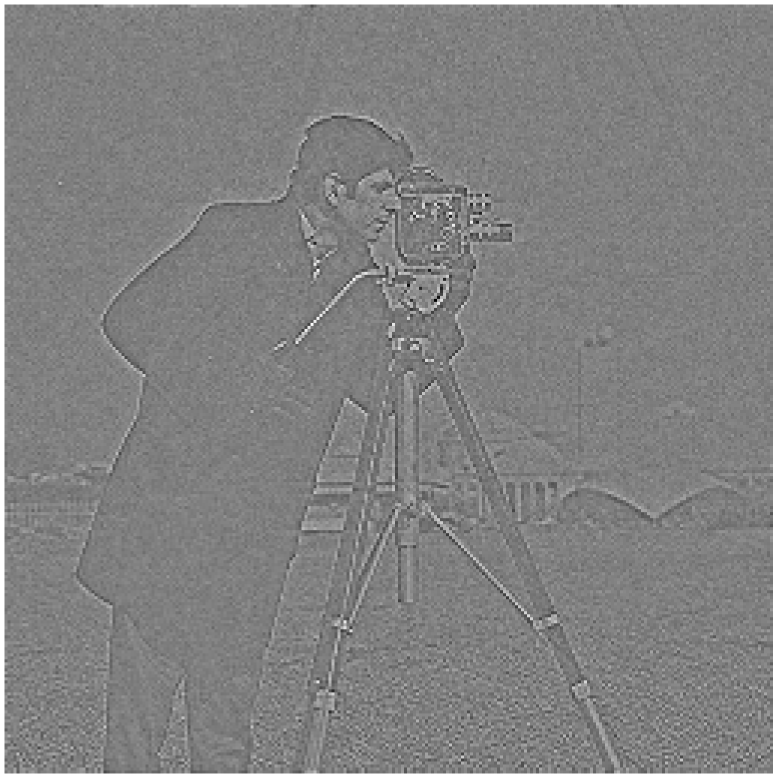


7.

$\mathbf{J}_1'$



$\mathbf{J}_2$



C program listing:

```
/*
 * hw04.c:
 *
 *  Solution for homework 4 (2D DFT).
 *
 *
 * 11/13/2000 jph
 *
 *   2/28/2007: changed FFT routine from numerical recipes to fftw3.  jph.
 *   3/30/2011: this is now HW 5 (used to be HW 6). jph.
 *   3/27/2013: LTI filtering problem moved to next HW assignment.  jph.
 *   2/15/2015: this is now HW 4. Added problems 6 and 7. 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  Problem1();
 void  Problem2();
 void  Problem3();
 void  Problem4();
 void  Problem5();
 void  Problem6();
 void  Problem7();
 BYTE *f2b_FullScale();
 void  WriteAsciiDFT();
 float L2Norm();
 void  fft2d();


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

main(argc,argv)

  int     argc;
  char   *argv[];
{

 /*------------------------------------------------------------------------
  * Call a function to do each problem
  */
  Problem1(argv[0]);
  Problem2(argv[0]);
  Problem3(argv[0]);
  Problem4(argv[0]);
  Problem5(argv[0]);
  Problem6(argv[0]);
  Problem7(argv[0]);

  return;

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

```
/*----------------------------------------------------------------------
 * Problem1
 *
 *   HW 4, Problem 1.
 *
 *
 *  jph 13 November 2000
 *
 -----------------------------------------------------------------------*/

void Problem1(cmd)

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

  int      size;                 /* num rows/cols in image */
  int      row;                  /* image row counter */
  int      col;                  /* image column counter */
  float   *I1R;                  /* Real part of image I1 */
  float   *I1I;                  /* Imaginary part of image I1 */
  BYTE    *BI1R;                 /* Real part of image I1 as BYTE image */
  BYTE    *BI1I;                 /* Imaginary part of image I1 as BYTE image */
  float   *Itilde1R;             /* Real part of DFT of image I1 */
  float   *Itilde1I;             /* Imaginary part of DFT of image I1 */
  double   u0;                   /* horizontal frequency of image */
  double   v0;                   /* vertical frequency of image */
  double   TwoPiOnSize;          /* 2 * pi / size */
  double   Phase;                /* argument of transcendentals */

 /*
  * Make the image I1.
  */
  /*****  constants  *****/
  size = 8;
  TwoPiOnSize = (double)2.0*M_PI / (double)size;
  u0 = v0 = (double)2.0;

  /*****  allocate image arrays  *****/
  if ((I1R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1.\n",cmd);
    exit(-1);
  }
  if ((I1I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1.\n",cmd);
    exit(-1);
  }
  if ((Itilde1R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1.\n",cmd);
    exit(-1);
  }
  if ((Itilde1I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 1.\n",cmd);
    exit(-1);
  }

  /*****  set the pixel values  *****/
  for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
      Phase = TwoPiOnSize * (u0*(double)col + v0*(double)row);
      I1R[row*size + col] = (float)0.5 * (float)cos(Phase);
      I1I[row*size + col] = (float)0.5 * (float)sin(Phase);
    }
  }

 /*
  * Make BYTE versions of I1R and I1I w/ full-scale contrast and output
  */
  BI1R = f2b_FullScale(I1R,size,cmd);
  BI1I = f2b_FullScale(I1I,size,cmd);
  byte2disk(BI1R,size,size,"I1R.bin");
  byte2disk(BI1I,size,size,"I1I.bin");

 /*
  * Compute the DFT of the image I1 and output
  */
  fft2d(I1R,I1I,Itilde1R,Itilde1I,size,1);
  WriteAsciiDFT(Itilde1R,size,"Itilde1R.ascii");
  WriteAsciiDFT(Itilde1I,size,"Itilde1I.ascii");

 /*
  * Clean up
  */
  free(I1R);
  free(I1I);
  free(Itilde1R);
  free(Itilde1I);
  free(BI1R);
  free(BI1I);

  return;
} /*---------------------  Problem1  ---------------------------------*/


/*-------------------------------------------------------------------
 * Problem2
 *
 *   HW 4, Problem 2.
 *
 *
 *  jph 13 November 2000
 *
 ---------------------------------------------------------------------*/

void Problem2(cmd)

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

  int      size;                  /* num rows/cols in image */
  int      row;                   /* image row counter */
  int      col;                   /* image column counter */
  float   *I2R;                   /* Real part of image I2 */
  float   *I2I;                   /* Imaginary part of image I2 */
  BYTE    *BI2R;                  /* Real part of image I2 as BYTE image */
  BYTE    *BI2I;                  /* Imaginary part of image I2 as BYTE image */
  float   *Itilde2R;              /* Real part of DFT of image I2 */
  float   *Itilde2I;              /* Imaginary part of DFT of image I2 */
  double   u0;                    /* horizontal frequency of image */
  double   v0;                    /* vertical frequency of image */
  double   TwoPiOnSize;           /* 2 * pi / size */
  double   Phase;                 /* argument of transcendentals */

 /*
  * Make the image I2.
  */
  /*****  constants  *****/
  size = 8;
  TwoPiOnSize = (double)2.0*M_PI / (double)size;
  u0 = v0 = (double)2.0;

  /*****  allocate image arrays  *****/
  if ((I2R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2.\n",cmd);
    exit(-1);
  }
  if ((I2I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2.\n",cmd);
    exit(-1);
  }
  if ((Itilde2R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2.\n",cmd);
    exit(-1);
  }
  if ((Itilde2I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 2.\n",cmd);
    exit(-1);
  }

  /*****  set the pixel values  *****/
  for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
      Phase = TwoPiOnSize * (u0*(double)col + v0*(double)row);
      I2R[row*size + col] = (float) 0.5 * (float)cos(Phase);
      I2I[row*size + col] = (float)-0.5 * (float)sin(Phase);
    }
  }

 /*
  * Make BYTE versions of I2R and I2I w/ full-scale contrast and output
  */
  BI2R = f2b_FullScale(I2R,size,cmd);
  BI2I = f2b_FullScale(I2I,size,cmd);
  byte2disk(BI2R,size,size,"I2R.bin");
  byte2disk(BI2I,size,size,"I2I.bin");

 /*
  * Compute the DFT of the image I2 and output
  */
  fft2d(I2R,I2I,Itilde2R,Itilde2I,size,1);
  WriteAsciiDFT(Itilde2R,size,"Itilde2R.ascii");
  WriteAsciiDFT(Itilde2I,size,"Itilde2I.ascii");

 /*
  * Clean up
  */
  free(I2R);
  free(I2I);
  free(Itilde2R);
  free(Itilde2I);
  free(BI2R);
  free(BI2I);

  return;
} /*---------------------  Problem2  ---------------------------------*/

/*----------------------------------------------------------------------
 * Problem3
 *
 *   HW 4, Problem 3.
 *
 *
 *  jph 13 November 2000
 *
 ----------------------------------------------------------------------*/

void Problem3(cmd)

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

  int      size;                  /* num rows/cols in image */
  int      row;                   /* image row counter */
  int      col;                   /* image column counter */
  float   *I3R;                   /* Real part of image I3 */
  float   *I3I;                   /* Imaginary part of image I3 */
  BYTE    *BI3R;                  /* Real part of image I3 as BYTE image */
  BYTE    *BI3I;                  /* Imaginary part of image I3 as BYTE image */
  float   *Itilde3R;              /* Real part of DFT of image I3 */
  float   *Itilde3I;              /* Imaginary part of DFT of image I3 */
  double   u0;                    /* horizontal frequency of image */
  double   v0;                    /* vertical frequency of image */
  double   TwoPiOnSize;           /* 2 * pi / size */
  double   Phase;                 /* argument of transcendentals */

 /*
  * Make the image I3.
  */
  /*****  constants  *****/
  size = 8;
  TwoPiOnSize = (double)2.0*M_PI / (double)size;
  u0 = v0 = (double)2.0;

  /*****  allocate image arrays  *****/
  if ((I3R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 3.\n",cmd);
    exit(-1);
  }
  if ((I3I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 3.\n",cmd);
    exit(-1);
  }
  if ((Itilde3R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 3.\n",cmd);
    exit(-1);
  }
  if ((Itilde3I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 3.\n",cmd);
    exit(-1);
  }

  /*****  set the pixel values  *****/
  for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
      Phase = TwoPiOnSize * (u0*(double)col + v0*(double)row);
      I3R[row*size + col] = (float)cos(Phase);
      I3I[row*size + col] = (float)0.0;
    }
  }

 /*
  * Make BYTE versions of I3R and I3I w/ full-scale contrast and output
  */
  BI3R = f2b_FullScale(I3R,size,cmd);
  BI3I = f2b_FullScale(I3I,size,cmd);
  byte2disk(BI3R,size,size,"I3R.bin");
  byte2disk(BI3I,size,size,"I3I.bin");

 /*
  * Compute the DFT of the image I3 and output
  */
  fft2d(I3R,I3I,Itilde3R,Itilde3I,size,1);
  WriteAsciiDFT(Itilde3R,size,"Itilde3R.ascii");
  WriteAsciiDFT(Itilde3I,size,"Itilde3I.ascii");

 /*
  * Clean up
  */
  free(I3R);
  free(I3I);
  free(Itilde3R);
  free(Itilde3I);
  free(BI3R);
  free(BI3I);

  return;
} /*---------------------  Problem3  ---------------------------------*/

/*------------------------------------------------------------------------
 * Problem4
 *
 *   HW 4, Problem 4.
 *
 *
 *  jph 13 November 2000
 *
 ------------------------------------------------------------------------*/

void Problem4(cmd)

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

  int      size;                  /* num rows/cols in image */
  int      row;                   /* image row counter */
  int      col;                   /* image column counter */
  float   *I4R;                   /* Real part of image I4 */
  float   *I4I;                   /* Imaginary part of image I4 */
  BYTE    *BI4R;                  /* Real part of image I4 as BYTE image */
  BYTE    *BI4I;                  /* Imaginary part of image I4 as BYTE image */
  float   *Itilde4R;              /* Real part of DFT of image I4 */
  float   *Itilde4I;              /* Imaginary part of DFT of image I4 */
  double   u0;                    /* horizontal frequency of image */
  double   v0;                    /* vertical frequency of image */
  double   TwoPiOnSize;           /* 2 * pi / size */
  double   Phase;                 /* argument of transcendentals */

 /*
  * Make the image I4.
  */
  /*****  constants  *****/
  size = 8;
  TwoPiOnSize = (double)2.0*M_PI / (double)size;
  u0 = v0 = (double)2.0;

  /*****  allocate image arrays  *****/
  if ((I4R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 4.\n",cmd);
    exit(-1);
  }
  if ((I4I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 4.\n",cmd);
    exit(-1);
  }
  if ((Itilde4R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 4.\n",cmd);
    exit(-1);
  }
  if ((Itilde4I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 4.\n",cmd);
    exit(-1);
  }

  /*****  set the pixel values  *****/
  for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
      Phase = TwoPiOnSize * (u0*(double)col + v0*(double)row);
      I4R[row*size + col] = (float)sin(Phase);
      I4I[row*size + col] = (float)0.0;
    }
  }

 /*
  * Make BYTE versions of I4R and I4I w/ full-scale contrast and output
  */
  BI4R = f2b_FullScale(I4R,size,cmd);
  BI4I = f2b_FullScale(I4I,size,cmd);
  byte2disk(BI4R,size,size,"I4R.bin");
  byte2disk(BI4I,size,size,"I4I.bin");

 /*
```

```
  * Compute the DFT of the image I4 and output
  */
  fft2d(I4R,I4I,Itilde4R,Itilde4I,size,1);
  WriteAsciiDFT(Itilde4R,size,"Itilde4R.ascii");
  WriteAsciiDFT(Itilde4I,size,"Itilde4I.ascii");

 /*
  * Clean up
  */
  free(I4R);
  free(I4I);
  free(Itilde4R);
  free(Itilde4I);
  free(BI4R);
  free(BI4I);

  return;
} /*---------------------  Problem4  ---------------------------------*/

/*----------------------------------------------------------------------
 * Problem5
 *
 *   HW 4, Problem 5.
 *
 *
 *  jph 13 November 2000
 *
 -----------------------------------------------------------------------*/

void Problem5(cmd)

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

  int      size;                  /* num rows/cols in image */
  int      row;                   /* image row counter */
  int      col;                   /* image column counter */
  float   *I5R;                   /* Real part of image I5 */
  float   *I5I;                   /* Imaginary part of image I5 */
  BYTE    *BI5R;                  /* Real part of image I5 as BYTE image */
  BYTE    *BI5I;                  /* Imaginary part of image I5 as BYTE image */
  float   *Itilde5R;              /* Real part of DFT of image I5 */
  float   *Itilde5I;              /* Imaginary part of DFT of image I5 */
  double   u1;                    /* horizontal frequency of image */
  double   v1;                    /* vertical frequency of image */
  double   TwoPiOnSize;           /* 2 * pi / size */
  double   Phase;                 /* argument of transcendentals */

 /*
  * Make the image I5.
  */
  /*****  constants  *****/
  size = 8;
  TwoPiOnSize = (double)2.0*M_PI / (double)size;
  u1 = v1 = (double)1.5;

  /*****  allocate image arrays  *****/
  if ((I5R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 5.\n",cmd);
    exit(-1);
  }
  if ((I5I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 5.\n",cmd);
    exit(-1);
  }
  if ((Itilde5R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 5.\n",cmd);
    exit(-1);
  }
  if ((Itilde5I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 5.\n",cmd);
    exit(-1);
  }

  /*****  set the pixel values  *****/
  for (row=0; row < size; row++) {
    for (col=0; col < size; col++) {
      Phase = TwoPiOnSize * (u1*(double)col + v1*(double)row);
      I5R[row*size + col] = (float)cos(Phase);
      I5I[row*size + col] = (float)0.0;
    }
  }

 /*
  * Make BYTE versions of I5R and I5I w/ full-scale contrast and output
  */
  BI5R = f2b_FullScale(I5R,size,cmd);
  BI5I = f2b_FullScale(I5I,size,cmd);
  byte2disk(BI5R,size,size,"I5R.bin");
  byte2disk(BI5I,size,size,"I5I.bin");

 /*
  * Compute the DFT of the image I5 and output
  */
  fft2d(I5R,I5I,Itilde5R,Itilde5I,size,1);
  WriteAsciiDFT(Itilde5R,size,"Itilde5R.ascii");
  WriteAsciiDFT(Itilde5I,size,"Itilde5I.ascii");

 /*
  * Clean up
  */
  free(I5R);
  free(I5I);
  free(Itilde5R);
  free(Itilde5I);
  free(BI5R);
  free(BI5I);

  return;
} /*---------------------  Problem5  ----------------------------------*/

/*----------------------------------------------------------------------
 * Problem6
 *
 *   HW 4, Problem 6.
 *
 *
 *  jph 15 February 2015
 *
 -----------------------------------------------------------------------*/

void Problem6(cmd)

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

  int      i;                     /* loop counter */
  int      size;                  /* num rows/cols in image */
  int      N;                     /* size * size */
  BYTE    *X;                     /* BYTE input image */
  float   *XR;                    /* Real part of image as float */
  float   *XI;                    /* Imag part of image as float (all zero) */
  float   *XtildeR;               /* Real part of DFT */
  float   *XtildeI;               /* Imag part of DFT */
  float   *XtildeMag;             /* DFT log-magnitude spectrum */
  float   *XtildePhase;           /* DFT phase */
  BYTE   *BXtildeR;               /* Real part of DFT as BYTE */
  BYTE   *BXtildeI;               /* Imag part of DFT as BYTE */
  BYTE   *BXtildeMag;             /* DFT log-magnitude spectrum as BYTE */
  BYTE   *BXtildePhase;           /* DFT phase as BYTE */
```

```
size = 256;
N = size * size;

/*****  allocate image arrays  *****/
if ((X = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((XR = (float*)malloc(size*size*sizeof(float))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((XI = (float*)malloc(size*size*sizeof(float))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((XtildeR = (float*)malloc(size*size*sizeof(float))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((XtildeI = (float*)malloc(size*size*sizeof(float))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((XtildeMag = (float*)malloc(size*size*sizeof(float))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((XtildePhase = (float*)malloc(size*size*sizeof(float))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((BXtildeR = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((BXtildeI = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((BXtildeMag = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}
if ((BXtildePhase = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
  printf("\n%s: free store exhausted in Problem 6.\n",cmd);
  exit(-1);
}

/************  CAMERAMAN  ******************************/
/* Read the input image, cast to float, and compute DFT */
/*sprintf(fn,"%s","camera.bin");*/
disk2byte(X,size,size,"camera.bin");
for (i=0; i < N; i++) {
  XR[i] = (float)(X[i]);
  XI[i] = (float)0.0;
}
fft2d(XR,XI,XtildeR,XtildeI,size,1);


/* Compute DFT log-magnitude spectrum and phase */
for (i=0; i < N; i++) {
  XtildeMag[i] = (float)log((double)1.0 + L2Norm(XtildeR[i],XtildeI[i]));
  XtildePhase[i] = (float)atan2((double)XtildeI[i],(double)XtildeR[i]);
}

/* Cast results to BYTE and write output files */
BXtildeR = f2b_FullScale(XtildeR,size,cmd);
BXtildeI = f2b_FullScale(XtildeI,size,cmd);
```

```
BXtildeMag = f2b_FullScale(XtildeMag,size,cmd);
BXtildePhase = f2b_FullScale(XtildePhase,size,cmd);
byte2disk(BXtildeR,size,size,"cameraFFTR.bin");
byte2disk(BXtildeI,size,size,"cameraFFTI.bin");
byte2disk(BXtildeMag,size,size,"cameraFFTMag.bin");
byte2disk(BXtildePhase,size,size,"cameraFFTPhase.bin");


/************  SALESMAN  *******************************/
/* Read the input image, cast to float, and compute DFT */
/*sprintf(fn,"%s","camera.bin");*/
disk2byte(X,size,size,"salesman.bin");
for (i=0; i < N; i++) {
  XR[i] = (float)(X[i]);
  XI[i] = (float)0.0;
}
fft2d(XR,XI,XtildeR,XtildeI,size,1);


/* Compute DFT log-magnitude spectrum and phase */
for (i=0; i < N; i++) {
  XtildeMag[i] = (float)log((double)1.0 + L2Norm(XtildeR[i],XtildeI[i]));
  XtildePhase[i] = (float)atan2((double)XtildeI[i],(double)XtildeR[i]);
}

/* Cast results to BYTE and write output files */
BXtildeR = f2b_FullScale(XtildeR,size,cmd);
BXtildeI = f2b_FullScale(XtildeI,size,cmd);
BXtildeMag = f2b_FullScale(XtildeMag,size,cmd);
BXtildePhase = f2b_FullScale(XtildePhase,size,cmd);
byte2disk(BXtildeR,size,size,"salesmanFFTR.bin");
byte2disk(BXtildeI,size,size,"salesmanFFTI.bin");
byte2disk(BXtildeMag,size,size,"salesmanFFTMag.bin");
byte2disk(BXtildePhase,size,size,"salesmanFFTPhase.bin");


/************  HEAD  ***********************************/
/* Read the input image, cast to float, and compute DFT */
/*sprintf(fn,"%s","camera.bin");*/
disk2byte(X,size,size,"head.bin");
for (i=0; i < N; i++) {
  XR[i] = (float)(X[i]);
  XI[i] = (float)0.0;
}
fft2d(XR,XI,XtildeR,XtildeI,size,1);


/* Compute DFT log-magnitude spectrum and phase */
for (i=0; i < N; i++) {
  XtildeMag[i] = (float)log((double)1.0 + L2Norm(XtildeR[i],XtildeI[i]));
  XtildePhase[i] = (float)atan2((double)XtildeI[i],(double)XtildeR[i]);
}

/* Cast results to BYTE and write output files */
BXtildeR = f2b_FullScale(XtildeR,size,cmd);
BXtildeI = f2b_FullScale(XtildeI,size,cmd);
BXtildeMag = f2b_FullScale(XtildeMag,size,cmd);
BXtildePhase = f2b_FullScale(XtildePhase,size,cmd);
byte2disk(BXtildeR,size,size,"headFFTR.bin");
byte2disk(BXtildeI,size,size,"headFFTI.bin");
byte2disk(BXtildeMag,size,size,"headFFTMag.bin");
byte2disk(BXtildePhase,size,size,"headFFTPhase.bin");


/************  EYER  ***********************************/
/* Read the input image, cast to float, and compute DFT */
/*sprintf(fn,"%s","camera.bin");*/
disk2byte(X,size,size,"eyeR.bin");
for (i=0; i < N; i++) {
  XR[i] = (float)(X[i]);
```

```
    XI[i] = (float)0.0;
  }
  fft2d(XR,XI,XtildeR,XtildeI,size,1);


  /* Compute DFT log-magnitude spectrum and phase */
  for (i=0; i < N; i++) {
    XtildeMag[i] = (float)log((double)1.0 + L2Norm(XtildeR[i],XtildeI[i]));
    XtildePhase[i] = (float)atan2((double)XtildeI[i],(double)XtildeR[i]);
  }

  /* Cast results to BYTE and write output files */
  BXtildeR = f2b_FullScale(XtildeR,size,cmd);
  BXtildeI = f2b_FullScale(XtildeI,size,cmd);
  BXtildeMag = f2b_FullScale(XtildeMag,size,cmd);
  BXtildePhase = f2b_FullScale(XtildePhase,size,cmd);
  byte2disk(BXtildeR,size,size,"eyeRFFTR.bin");
  byte2disk(BXtildeI,size,size,"eyeRFFTI.bin");
  byte2disk(BXtildeMag,size,size,"eyeRFFTMag.bin");
  byte2disk(BXtildePhase,size,size,"eyeRFFTPhase.bin");

 /*
  * Clean up
  */
  free(X);
  free(XR);
  free(XI);
  free(XtildeR);
  free(XtildeI);
  free(XtildeMag);
  free(XtildePhase);
  free(BXtildeR);
  free(BXtildeI);
  free(BXtildeMag);
  free(BXtildePhase);


  return;
} /*---------------------  Problem6  ---------------------------------*/

/*----------------------------------------------------------------------
 * Problem7
 *
 *   HW 4, Problem 7.
 *
 *
 *  jph 15 February 2015
 *
 ----------------------------------------------------------------------*/

void Problem7(cmd)

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

  int      i;                    /* loop counter */
  int      size;                 /* num rows/cols in image */
  int      N;                    /* size * size */
  BYTE    *X;                    /* BYTE input image */
  float   *XR;                   /* Real part of image as float */
  float   *XI;                   /* Imag part of image as float (all zero) */
  float   *XtildeR;              /* Real part of DFT */
  float   *XtildeI;              /* Imag part of DFT */
  float   *XtildeMag;            /* DFT magnitude */
  float   *XtildePhase;          /* DFT phase */
  BYTE    *J1;                   /* BYTE J1 image */
  float   *J1R;                  /* Real part of J1 image as float */
  float   *J1I;                  /* Imag part of J1 image as float (all zero) */
  float   *J1tildeR;             /* Real part of J1 DFT */
  float   *J1tildeI;             /* Imag part of J1 DFT */
  float   *J1tildeMag;            /* J1 DFT magnitude */
  float   *J1tildePhase;          /* J1 DFT phase */
  BYTE    *J2;                    /* BYTE J2 image */
  float   *J2R;                   /* Real part of J2 image as float */
  float   *J2I;                   /* Imag part of J2 image as float (all zero) */
  float   *J2tildeR;              /* Real part of J2 DFT */
  float   *J2tildeI;              /* Imag part of J2 DFT */
  float   *J2tildeMag;            /* J2 DFT magnitude */
  float   *J2tildePhase;          /* J2 DFT phase */


  size = 256;
  N = size * size;

  /*****  allocate image arrays  *****/
  if ((X = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((XR = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((XI = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((XtildeR = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((XtildeI = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((XtildeMag = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((XtildePhase = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1 = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1tildeR = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1tildeI = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1tildeMag = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J1tildePhase = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2 = (BYTE*)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2R = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2I = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2tildeR = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2tildeI = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2tildeMag = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }
  if ((J2tildePhase = (float*)malloc(size*size*sizeof(float))) == NULL) {
    printf("\n%s: free store exhausted in Problem 6.\n",cmd);
    exit(-1);
  }

  /* Read the input image, cast to float, and compute DFT */
  /*sprintf(fn,"%s","camera.bin");*/
  disk2byte(X,size,size,"camera.bin");
  for (i=0; i < N; i++) {
    XR[i] = (float)(X[i]);
    XI[i] = (float)0.0;
  }
  fft2d(XR,XI,XtildeR,XtildeI,size,1);

  /* Compute DFT magnitude and phase */
  for (i=0; i < N; i++) {
    XtildeMag[i] = L2Norm(XtildeR[i],XtildeI[i]);
    XtildePhase[i] = (float)atan2((double)XtildeI[i],(double)XtildeR[i]);
  }

  /* Compute DFTs of J1 and J2; invert to get the images */
  for (i=0; i < N; i++) {
    J1tildeMag[i] = XtildeMag[i];
    J1tildePhase[i] = (float)0.0;
    J1tildeR[i] = J1tildeMag[i]*(float)cos((double)J1tildePhase[i]);
    J1tildeI[i] = J1tildeMag[i]*(float)sin((double)J1tildePhase[i]);

    J2tildeMag[i] = (float)1.0;
    J2tildePhase[i] = XtildePhase[i];
    J2tildeR[i] = J2tildeMag[i]*(float)cos((double)J2tildePhase[i]);
    J2tildeI[i] = J2tildeMag[i]*(float)sin((double)J2tildePhase[i]);
  }
  fft2d(J1R,J1I,J1tildeR,J1tildeI,size,-1);
  fft2d(J2R,J2I,J2tildeR,J2tildeI,size,-1);

  /* Do log range compression on J1 */
  for (i=0; i < N; i++) {
    J1R[i] = (float)log((double)J1R[i]);
  }

  /* Write results to files */
  J1 = f2b_FullScale(J1R,size,cmd);
  J2 = f2b_FullScale(J2R,size,cmd);
  byte2disk(J1,size,size,"J1prime.bin");
  byte2disk(J2,size,size,"J2.bin");
```

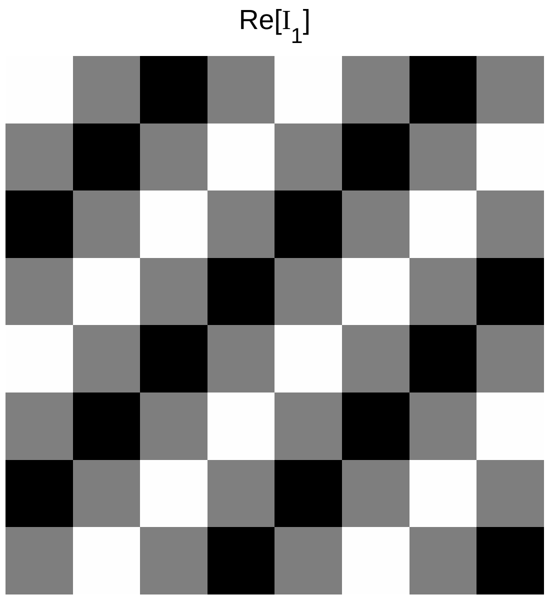
```
 /*
  * Clean up
  */
  free(X);
  free(XR);
  free(XI);
  free(XtildeR);
  free(XtildeI);
  free(XtildeMag);
  free(XtildePhase);
  free(J1);
  free(J1R);
  free(J1I);
  free(J1tildeR);
  free(J1tildeI);
  free(J1tildeMag);
  free(J1tildePhase);
  free(J2);
  free(J2R);
  free(J2I);
  free(J2tildeR);
  free(J2tildeI);
  free(J2tildeMag);
  free(J2tildePhase);

  return;
} /*---------------------  Problem7  ----------------------------------*/


/*----------------------------------------------------------------------
 * 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  -----------------------------*/

/*----------------------------------------------------------------------
 * WriteAsciiDFT
 *
 *   Write the Real or Imaginary part of an 8x8 DFT to an ascii disk file.
 *
 *
 *  jph 13 November 2000
 *
 ----------------------------------------------------------------------*/

void WriteAsciiDFT(x,size,OutFile)

  float   *x;                     /* input float DFT array */
  int      size;                  /* num rows/cols in image, assumed = 8 */
  char    *OutFile;               /* output filename */
{

  int      row;                   /* row counter */
  int      col;                   /* column counter */
  FILE    *OutFp;                 /* output file pointer */

 /*
  * Open the output file
  */
  if ((OutFp = fopen(OutFile,"w")) == NULL) {
    printf("\nWriteAsciiDFT: cannot open file %s\n",OutFile);
    exit(-1);
  }

 /*
  * Loop over rows and cols, write ascii data to file
  */
  for (row=0; row < size; row++) {
    fprintf(OutFp,"%9.4f",x[row*size + 0]);  /* 1st entry; no leading blank */
    for (col=1; col < size; col++) {
      fprintf(OutFp,"%10.4f",x[row*size + col]); /* rest w/ leading blank */
    }
    fprintf(OutFp,"\n");        /* eol */
  }

  fclose(OutFp);
  return;

} /*---------------------  WriteAsciiDFT  -----------------------------*/

/*----------------------------------------------------------------------
 * 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 -------------------------------------*/

/*--------------------------------------------------------------------
 * 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;
}

/*--------------------------------------------------------------------
 * 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  --------------------------------*/

/*
 * 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 */
```

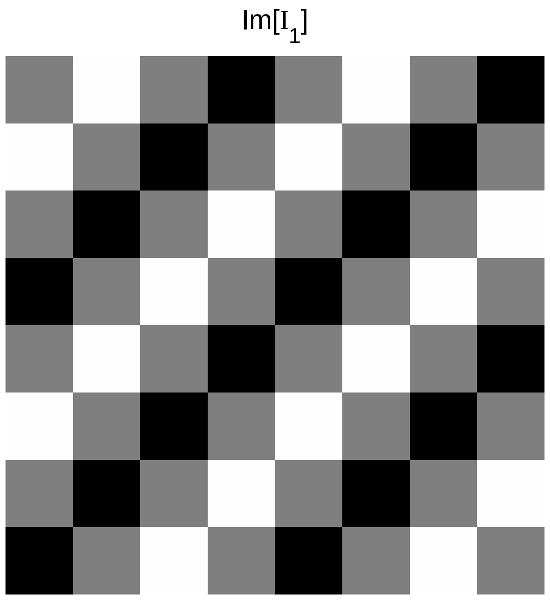
```
{
    fftw_complex *pass;           /* array for passing data to/from fftw */
    fftw_plan Plan;               /* structure for fftw3 planning */
    float one_on_n;               /* 1 / (no. data points in signal) */
    int i;                        /* counter */
    int n;                   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

Matlab Solution:

Note: Explanations are not given in the Matlab solution. The explanations are the same as the ones already given above in the C solution.

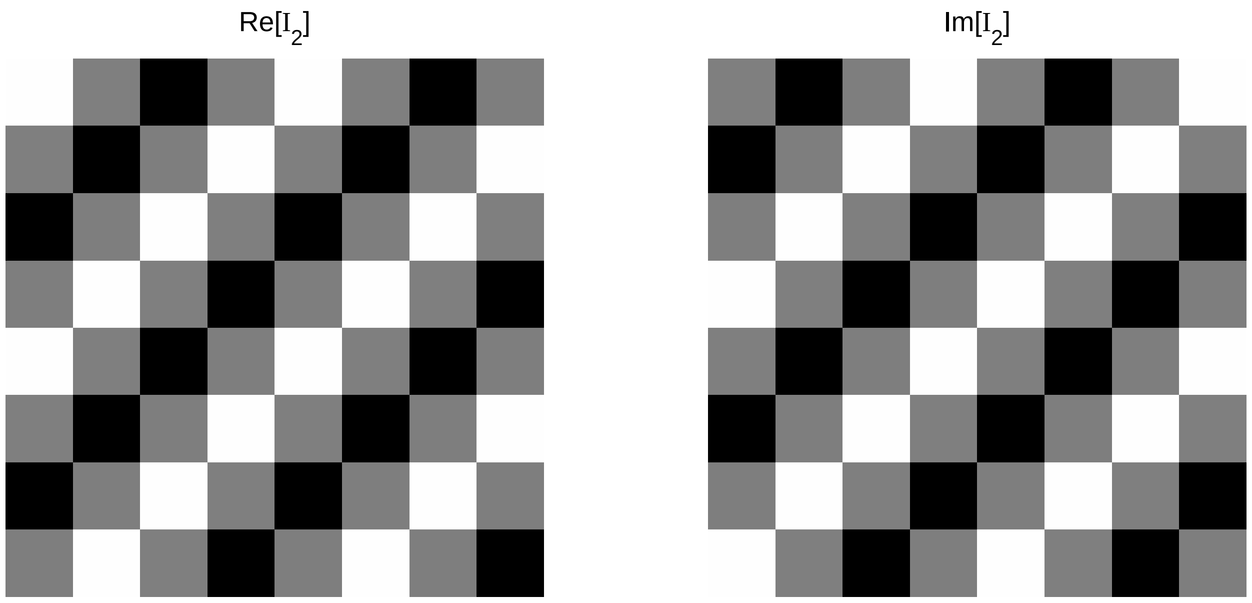
1.





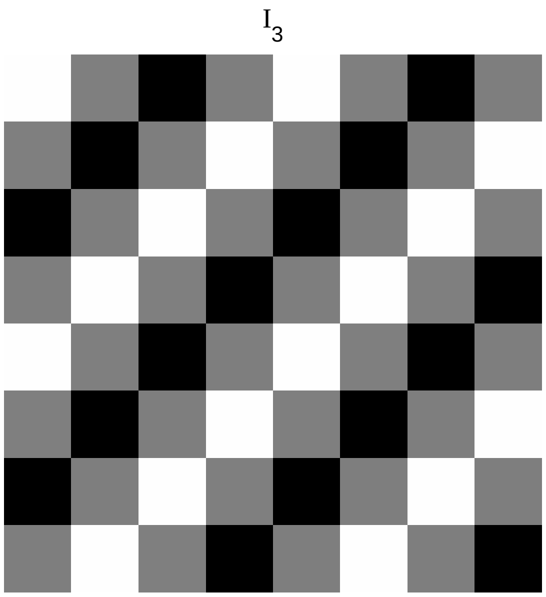
```
Re[DFT(I1)]:
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0    32     0
      0     0     0     0     0     0     0     0

Im[DFT(I1)]:
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
```

2.



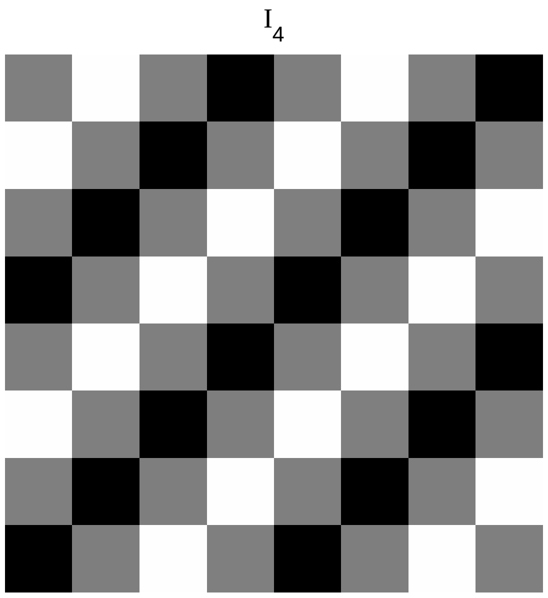
```
Re[DFT(I2)]:
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0      32       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0

Im[DFT(I2)]:
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
```

3.



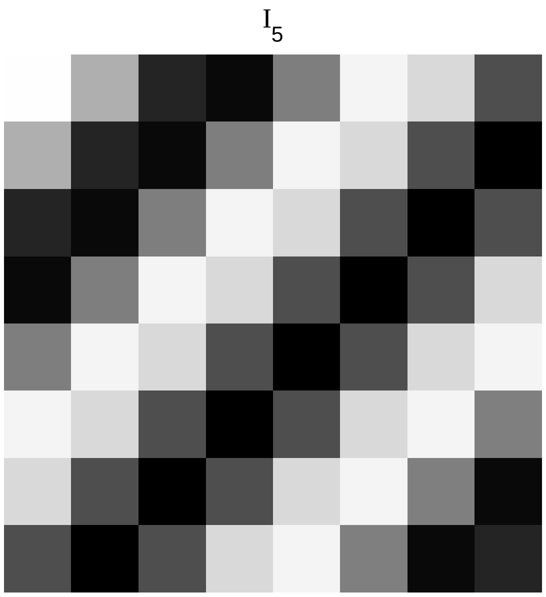
```
Re[DFT(I3)]:
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0     32      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0     32      0
      0      0      0      0      0      0      0      0

Im[DFT(I3)]:
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
```

4.



```
Re[DFT(I4)]:
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0

Im[DFT(I4)]:
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0     32      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0    -32      0
      0      0      0      0      0      0      0      0
```

5.

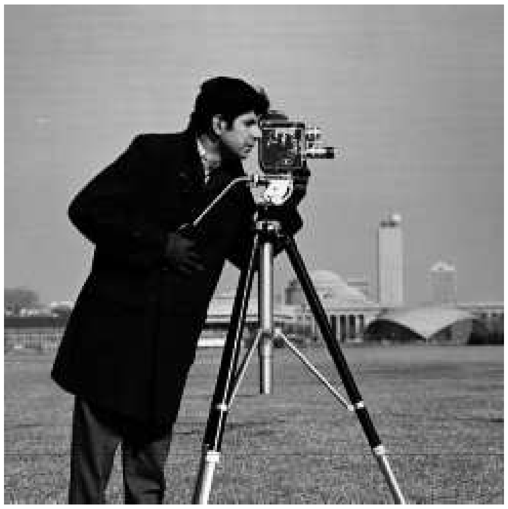


```
Re[DFT(I5)]:
    0.5535    0.4335   -0.6131    2.9028    2.0000    2.9028   -0.6131    0.4335
    0.4335   -0.1397   -2.7422    4.8284    2.2688    2.0000    0.6488    0.7023
   -0.6131   -2.7422  -11.6569   13.5706    4.6131    2.1796    2.0000    0.6488
    2.9028    4.8284   13.5706  -11.8603   -3.2620   -2.3592    2.1796    2.0000
    2.0000    2.2688    4.6131   -3.2620   -1.2398   -3.2620    4.6131    2.2688
    2.9028    2.0000    2.1796   -2.3592   -3.2620  -11.8603   13.5706    4.8284
   -0.6131    0.6488    2.0000    2.1796    4.6131   13.5706  -11.6569   -2.7422
    0.4335    0.7023    0.6488    2.0000    2.2688    4.8284   -2.7422   -0.1397

Im[DFT(I5)]:
         0    0.6488    2.6131   -2.1796         0    2.1796   -2.6131   -0.6488
    0.6488    1.2977    3.2620   -1.5307    0.6488    2.8284   -1.9643         0
    2.6131    3.2620    5.2263    0.4335    2.6131    4.7927         0    1.9643
   -2.1796   -1.5307    0.4335   -4.3592   -2.1796         0   -4.7927   -2.8284
         0    0.6488    2.6131   -2.1796         0    2.1796   -2.6131   -0.6488
    2.1796    2.8284    4.7927         0    2.1796    4.3592   -0.4335    1.5307
   -2.6131   -1.9643         0   -4.7927   -2.6131   -0.4335   -5.2263   -3.2620
   -0.6488         0    1.9643   -2.8284   -0.6488    1.5307   -3.2620   -1.2977
```

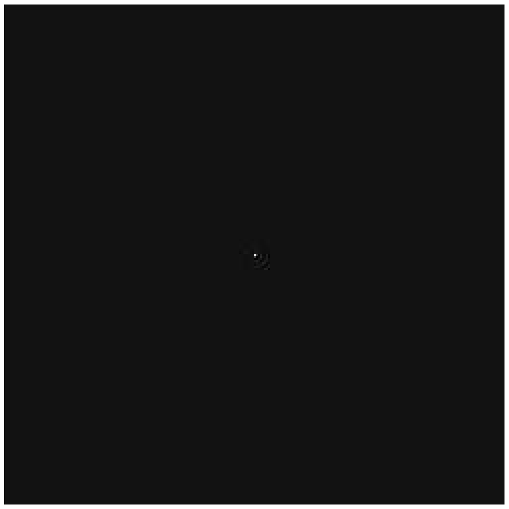
Discussion: The discussion is given in the C solution.

6.

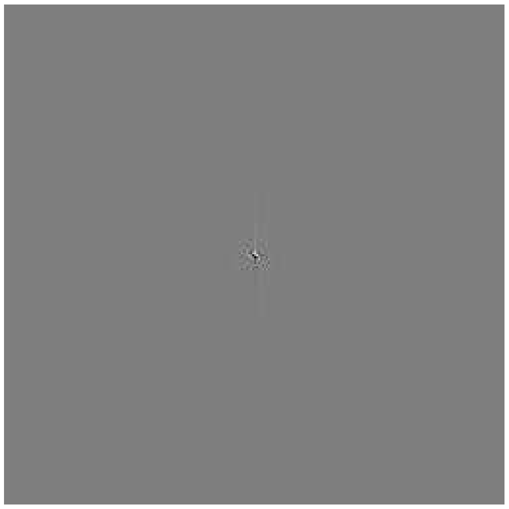
camera



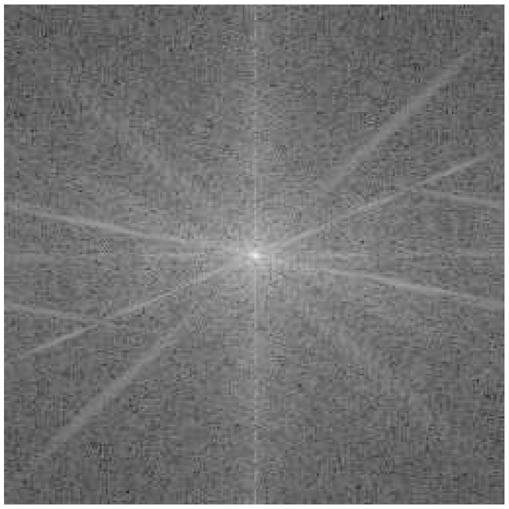
Re[DFT(camera)]



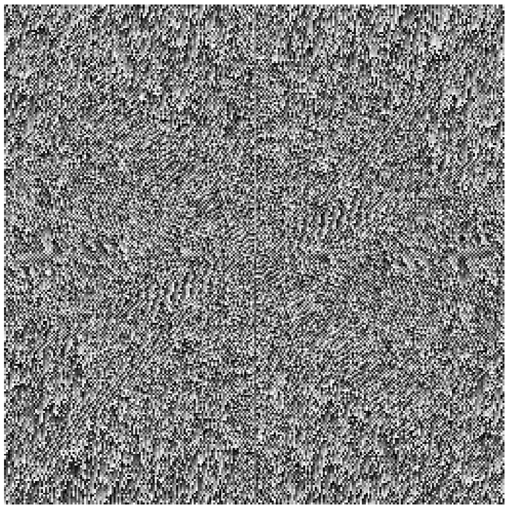
Im[DFT(camera)]



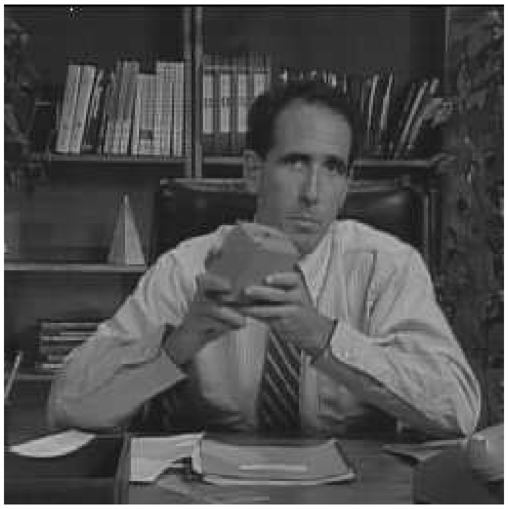
camera log magnitude spectrum



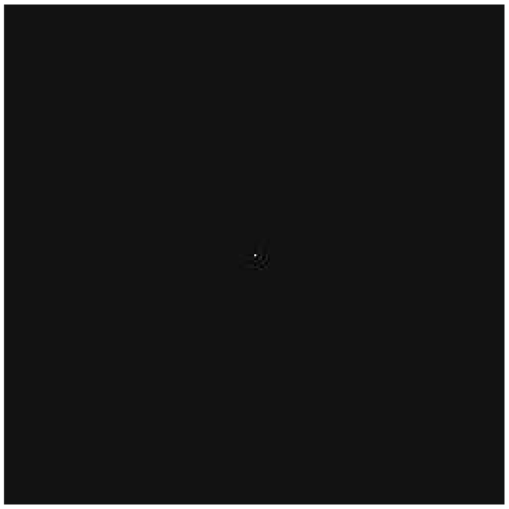
arg[DFT(camera)]



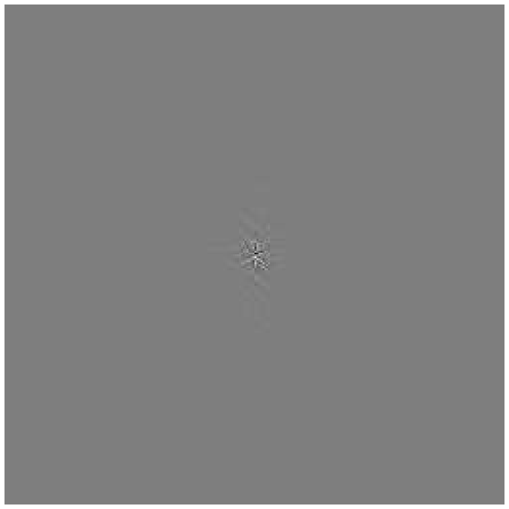
salesman



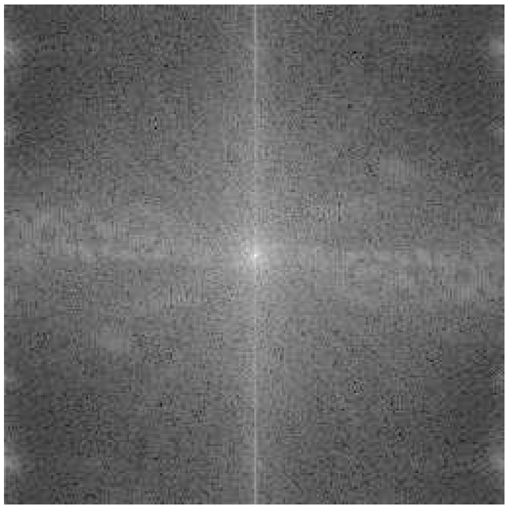
Re[DFT(salesman)]



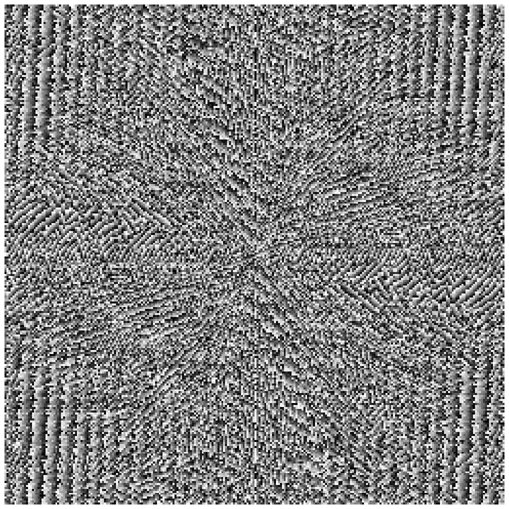
Im[DFT(salesman)]



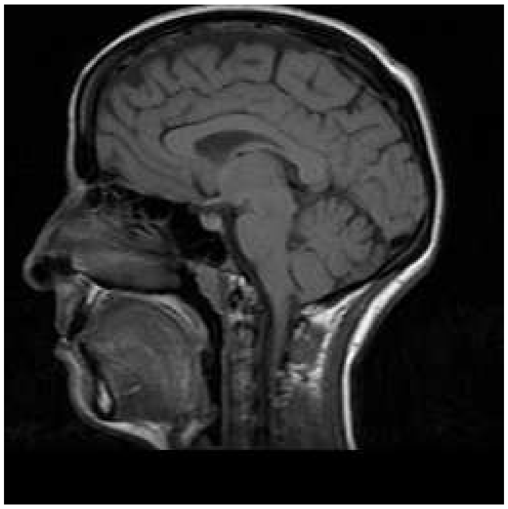
salesman log magnitude spectrum



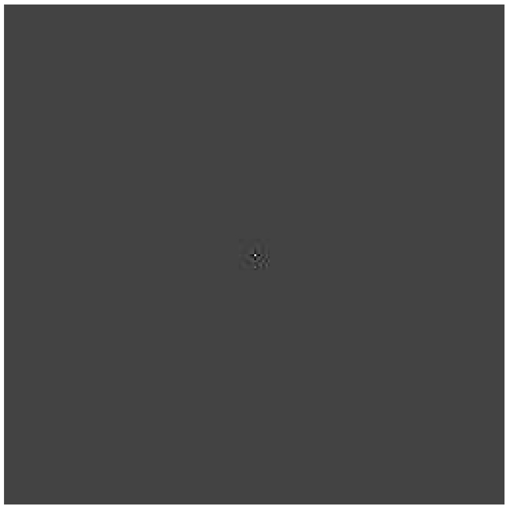
arg[DFT(salesman)]



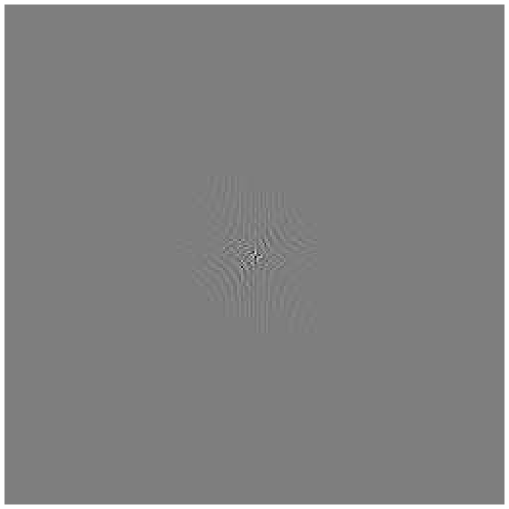
head



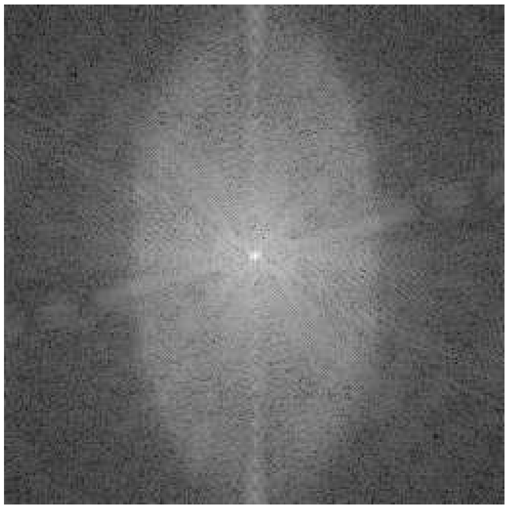
Re[DFT(head)]



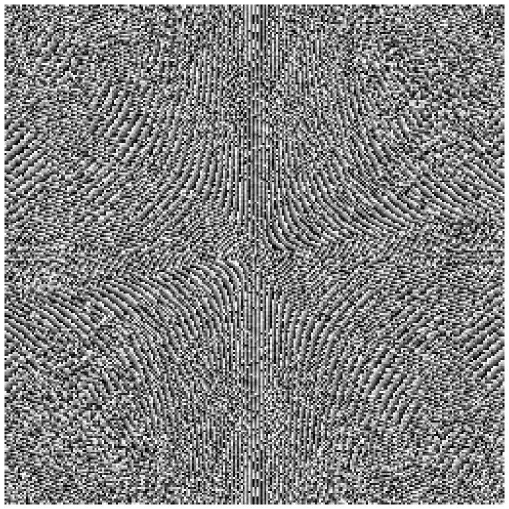
Im[DFT(head)]



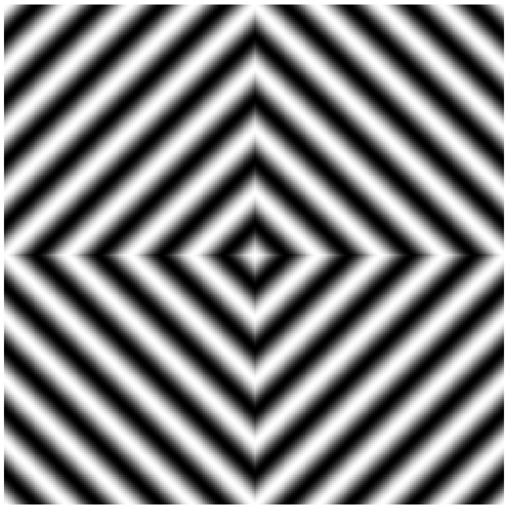
head log magnitude spectrum



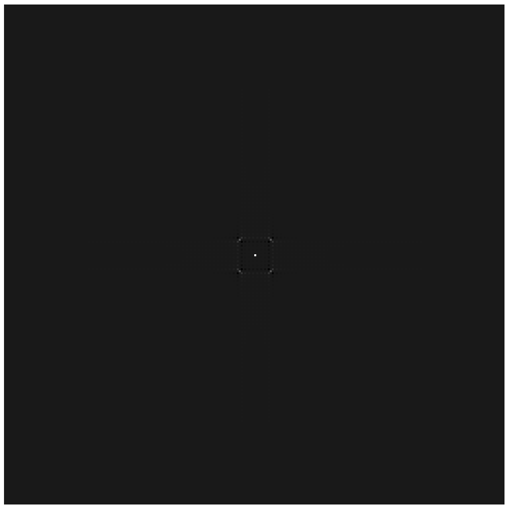
arg[DFT(head)]



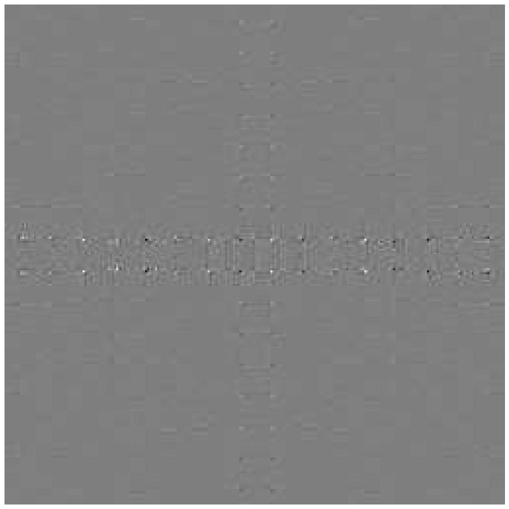
eyeR



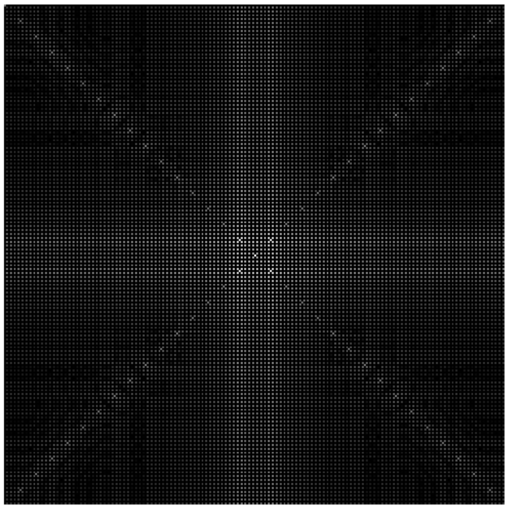
Re[DFT(eyeR)]



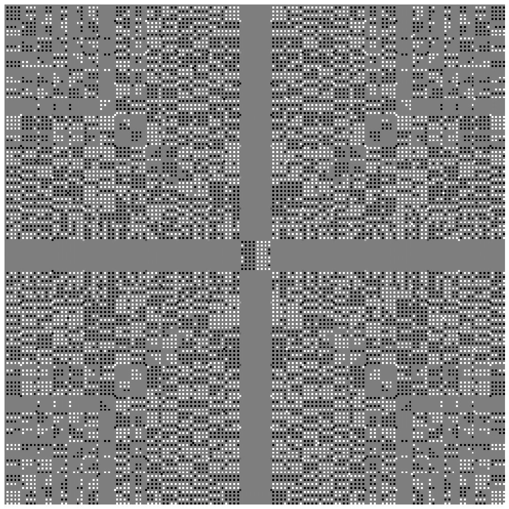
Im[DFT(eyeR)]



eyeR log magnitude spectrum

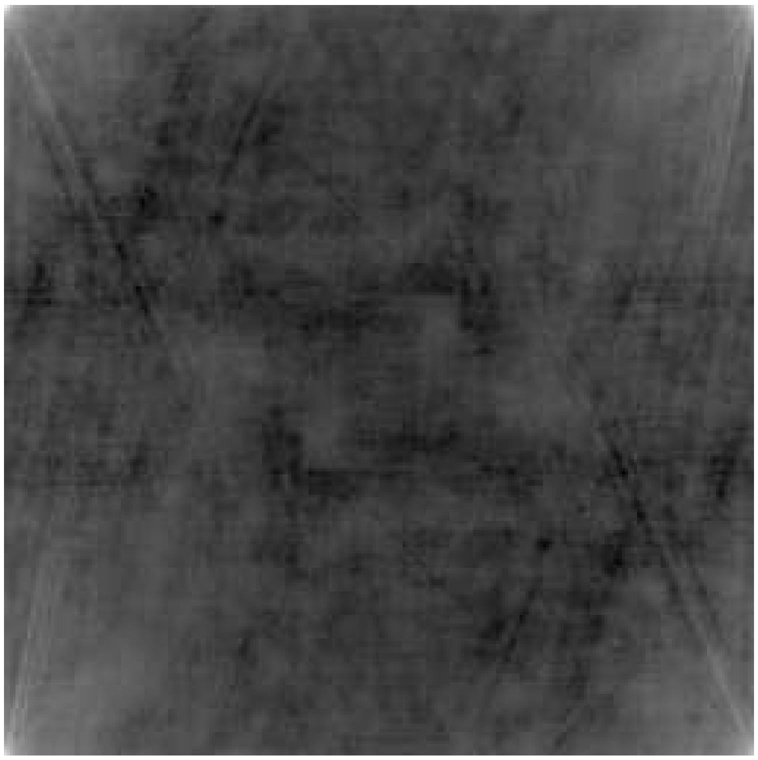


arg[DFT(eyeR)]



7.

J1’



J2

Matlab m-file listing:

```
%
% hw04.m
%
% 04/05/04 jph
%  03/27/2013 the LTI filtering problem is now postponed to a later homework
%  02/15/2015 added problems 6 and 7. jph.
%

diary hw04.diary

%--------------------------------------------------------------------------
% Problem 1
%
% Here the images are of size 8x8.  For creating these images, it is
% important that the columns and rows use zero-based indexing.
%
% Make an 8x8 matrix COLS where each entry gives the zero-based index of the
% column and an 8x8 matrix ROWS where each entry gives the zero-based index
% of the row.
%
[COLS,ROWS] = meshgrid(0:7,0:7);

%
% Make the image I1.  Display the real and imaginary parts as 8 bpp images
% with full-scale contrast.
%
u0 = 2.0;
v0 = 2.0;
I1 = 0.5 * exp(j*2*pi/8*(u0*COLS + v0*ROWS));

%
% Perform full-scale contrast stretch on real and imaginary parts of I1 and
%   display as images.
%
I1R = stretch(real(I1));
I1I = stretch(imag(I1));
figure(1);
image(I1R);colormap(gray(256));axis('image','off');
title('Re[{\fontname{times}I}_1]','FontSize',18);
print -deps MI1R.eps;
print -dpdf MI1R.pdf;

figure(2);
image(I1I);colormap(gray(256));axis('image','off');
title('Im[{\fontname{times}I}_1]','FontSize',18);
print -deps MI1I.eps;
print -dpdf MI1I.pdf;

%
% Compute the centered 2D DFT of I1.  Print out real and imaginary parts
% as ascii arrays.
%
Itilde1 = fftshift(fft2(I1));
fprintf(1,'%s\n','Re[DFT(I1)]:');
disp(round(real(Itilde1) * 10^4)*10^(-4));
fprintf(1,'%s\n','Im[DFT(I1)]:');
disp(round(imag(Itilde1) * 10^4)*10^(-4));

%--------------------------------------------------------------------------
% Problem 2
%

%
% Make the image I2.
%
I2 = 0.5 * exp(-j*2*pi/8*(u0*COLS + v0*ROWS));

%
```

```
% Display real and imaginary parts as 8 bpp images with full scale contrast.
%
I2R = stretch(real(I2));
I2I = stretch(imag(I2));

figure(3);
image(I2R);colormap(gray(256));axis('image','off');
title('Re[{\fontname{times}I}_2]','FontSize',18);
print -deps MI2R.eps;
print -dpdf MI2R.pdf;

figure(4);
image(I2I);colormap(gray(256));axis('image','off');
title('Im[{\fontname{times}I}_2]','FontSize',18);
print -deps MI2I.eps;
print -dpdf MI2I.pdf;

%
% Compute the centered 2D DFT of I2.  Print out real and imaginary parts
% as ascii arrays.
%
Itilde2 = fftshift(fft2(I2));
fprintf(1,'%s\n','Re[DFT(I2)]:');
disp(round(real(Itilde2) * 10^4)*10^(-4));
fprintf(1,'%s\n','Im[DFT(I2)]:');
disp(round(imag(Itilde2) * 10^4)*10^(-4));

%-------------------------------------------------------------------------
% Problem 3
%

%
% Make the image I3.
%
I3 = cos(2*pi/8*(u0*COLS + v0*ROWS));

%
% Display as 8 bpp image with full scale contrast.
%
figure(5);
image(stretch(I3));colormap(gray(256));axis('image','off');
title('{\fontname{times}I}_3','FontSize',18);
print -deps MI3R.eps;
print -dpdf MI3R.pdf;

%
% Compute the centered 2D DFT of I3.  Print out real and imaginary parts
% as ascii arrays.
%
Itilde3 = fftshift(fft2(I3));
fprintf(1,'%s\n','Re[DFT(I3)]:');
disp(round(real(Itilde3) * 10^4)*10^(-4));
fprintf(1,'%s\n','Im[DFT(I3)]:');
disp(round(imag(Itilde3) * 10^4)*10^(-4));

%-------------------------------------------------------------------------
% Problem 4
%

%
% Make the image I4.
%
I4 = sin(2*pi/8*(u0*COLS + v0*ROWS));

%
% Display as 8 bpp image with full scale contrast.
%
figure(6);
image(stretch(I4));colormap(gray(256));axis('image','off');
title('{\fontname{times}I}_4','FontSize',18);
```

```
print -deps MI4R.eps;
print -dpdf MI4R.pdf;

%
% Compute the centered 2D DFT of I4.  Print out real and imaginary parts
% as ascii arrays.
%
Itilde4 = fftshift(fft2(I4));
fprintf(1,'%s\n','Re[DFT(I4)]:');
disp(round(real(Itilde4) * 10^4)*10^(-4));
fprintf(1,'%s\n','Im[DFT(I4)]:');
disp(round(imag(Itilde4) * 10^4)*10^(-4));

%---------------------------------------------------------------------------
% Problem 5
%

%
% Make the image I5.
%
u1 = 1.5;
v1 = 1.5;
I5 = cos(2*pi/8*(u1*COLS + v1*ROWS));

%
% Display as 8 bpp image with full scale contrast.
%
figure(7);
image(stretch(I5));colormap(gray(256));axis('image','off');
title('{\fontname{times}I}_5','FontSize',18);
print -deps MI5R.eps;
print -dpdf MI5R.pdf;

%
% Compute the centered 2D DFT of I5.  Print out real and imaginary parts
% as ascii arrays.
%
Itilde5 = fftshift(fft2(I5));
fprintf(1,'%s\n','Re[DFT(I5)]:');
disp(round(real(Itilde5) * 10^4)*10^(-4));
fprintf(1,'%s\n','Im[DFT(I5)]:');
disp(round(imag(Itilde5) * 10^4)*10^(-4));

diary off

%---------------------------------------------------------------------------
% Problem 6
%

%
% Cameraman
%
X = ReadBin('camera.bin',256);
Xtilde = fftshift(fft2(X));
XtildeR = stretch(real(Xtilde));
XtildeI = stretch(imag(Xtilde));
XtildeMag = stretch(log(1 + abs(Xtilde)));
XtildePhase = stretch(angle(Xtilde));

figure(8);
image(X);colormap(gray(256));axis('image','off');
title('{\fontname{times}camera}','FontSize',18);
print -deps Mcamera.eps;
print -dpdf Mcamera.pdf;

figure(9);
image(XtildeR);colormap(gray(256));axis('image','off');
title('{\fontname{times}Re[DFT(camera)]}','FontSize',18);
print -deps McameraFFTR.eps;
print -dpdf McameraFFTR.pdf;
```

```
figure(10);
image(XtildeI);colormap(gray(256));axis('image','off');
title('{\fontname{times}Im[DFT(camera)]}','FontSize',18);
print -deps McameraFFTI.eps;
print -dpdf McameraFFTI.pdf;

figure(11);
image(XtildeMag);colormap(gray(256));axis('image','off');
title('{\fontname{times}camera log magnitude spectrum}','FontSize',18);
print -deps McameraFFTMag.eps;
print -dpdf McameraFFTMag.pdf;

figure(12);
image(XtildePhase);colormap(gray(256));axis('image','off');
title('{\fontname{times}arg[DFT(camera)]}','FontSize',18);
print -deps McameraFFTPhase.eps;
print -dpdf McameraFFTPhase.pdf;

%
% Salesman
%
X = ReadBin('salesman.bin',256);
Xtilde = fftshift(fft2(X));
XtildeR = stretch(real(Xtilde));
XtildeI = stretch(imag(Xtilde));
XtildeMag = stretch(log(1 + abs(Xtilde)));
XtildePhase = stretch(angle(Xtilde));

figure(13);
image(X);colormap(gray(256));axis('image','off');
title('{\fontname{times}salesman}','FontSize',18);
print -deps Msalesman.eps;
print -dpdf Msalesman.pdf;

figure(14);
image(XtildeR);colormap(gray(256));axis('image','off');
title('{\fontname{times}Re[DFT(salesman)]}','FontSize',18);
print -deps MsalesmanFFTR.eps;
print -dpdf MsalesmanFFTR.pdf;

figure(15);
image(XtildeI);colormap(gray(256));axis('image','off');
title('{\fontname{times}Im[DFT(salesman)]}','FontSize',18);
print -deps MsalesmanFFTI.eps;
print -dpdf MsalesmanFFTI.pdf;

figure(16);
image(XtildeMag);colormap(gray(256));axis('image','off');
title('{\fontname{times}salesman log magnitude spectrum}','FontSize',18);
print -deps MsalesmanFFTMag.eps;
print -dpdf MsalesmanFFTMag.pdf;

figure(17);
image(XtildePhase);colormap(gray(256));axis('image','off');
title('{\fontname{times}arg[DFT(salesman)]}','FontSize',18);
print -deps MsalesmanFFTPhase.eps;
print -dpdf MsalesmanFFTPhase.pdf;

%
% Head
%
X = ReadBin('head.bin',256);
Xtilde = fftshift(fft2(X));
XtildeR = stretch(real(Xtilde));
XtildeI = stretch(imag(Xtilde));
XtildeMag = stretch(log(1 + abs(Xtilde)));
XtildePhase = stretch(angle(Xtilde));

figure(18);
```

```
image(X);colormap(gray(256));axis('image','off');
title('{\fontname{times}head}','FontSize',18);
print -deps Mhead.eps;
print -dpdf Mhead.pdf;

figure(19);
image(XtildeR);colormap(gray(256));axis('image','off');
title('{\fontname{times}Re[DFT(head)]}','FontSize',18);
print -deps MheadFFTR.eps;
print -dpdf MheadFFTR.pdf;

figure(20);
image(XtildeI);colormap(gray(256));axis('image','off');
title('{\fontname{times}Im[DFT(head)]}','FontSize',18);
print -deps MheadFFTI.eps;
print -dpdf MheadFFTI.pdf;

figure(21);
image(XtildeMag);colormap(gray(256));axis('image','off');
title('{\fontname{times}head log magnitude spectrum}','FontSize',18);
print -deps MheadFFTMag.eps;
print -dpdf MheadFFTMag.pdf;

figure(22);
image(XtildePhase);colormap(gray(256));axis('image','off');
title('{\fontname{times}arg[DFT(head)]}','FontSize',18);
print -deps MheadFFTPhase.eps;
print -dpdf MheadFFTPhase.pdf;

%
% eyeR
%
X = ReadBin('eyeR.bin',256);
Xtilde = fftshift(fft2(X));
XtildeR = stretch(real(Xtilde));
XtildeI = stretch(imag(Xtilde));
XtildeMag = stretch(log(1 + abs(Xtilde)));
XtildePhase = stretch(angle(Xtilde));

figure(23);
image(X);colormap(gray(256));axis('image','off');
title('{\fontname{times}eyeR}','FontSize',18);
print -deps MeyeR.eps;
print -dpdf MeyeR.pdf;

figure(24);
image(XtildeR);colormap(gray(256));axis('image','off');
title('{\fontname{times}Re[DFT(eyeR)]}','FontSize',18);
print -deps MeyeRFFTR.eps;
print -dpdf MeyeRFFTR.pdf;

figure(25);
image(XtildeI);colormap(gray(256));axis('image','off');
title('{\fontname{times}Im[DFT(eyeR)]}','FontSize',18);
print -deps MeyeRFFTI.eps;
print -dpdf MeyeRFFTI.pdf;

figure(26);
image(XtildeMag);colormap(gray(256));axis('image','off');
title('{\fontname{times}eyeR log magnitude spectrum}','FontSize',18);
print -deps MeyeRFFTMag.eps;
print -dpdf MeyeRFFTMag.pdf;

figure(27);
image(XtildePhase);colormap(gray(256));axis('image','off');
title('{\fontname{times}arg[DFT(eyeR)]}','FontSize',18);
print -deps MeyeRFFTPhase.eps;
print -dpdf MeyeRFFTPhase.pdf;

%-----------------------------------------------------------------------------
```

```
% Problem 7
%

X = ReadBin('camera.bin',256);
Xtilde = fft2(X);
%
% since cos(0)=1 and sin(0)=0, Re[J1tilde] = |Xtilde|
%    and Im[J1tilde] = 0.
%
J1tilde = abs(Xtilde);
J1 = real(ifft2(J1tilde));
J1prime = stretch(log(1 + J1));
figure(28);
image(J1prime);colormap(gray(256));axis('image','off');
title('{\fontname{times}J1''}','FontSize',18);
print -deps MJ1prime.eps;
print -dpdf MJ1prime.pdf;

J2tilde = exp(j*angle(Xtilde));
J2 = stretch(real(ifft2(J2tilde)));
figure(29);
image(J2);colormap(gray(256));axis('image','off');
title('{\fontname{times}J2}','FontSize',18);
print -deps MJ2.eps;
print -dpdf MJ2.pdf;


%
% 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);
```