ECE 5273
FFT Handout

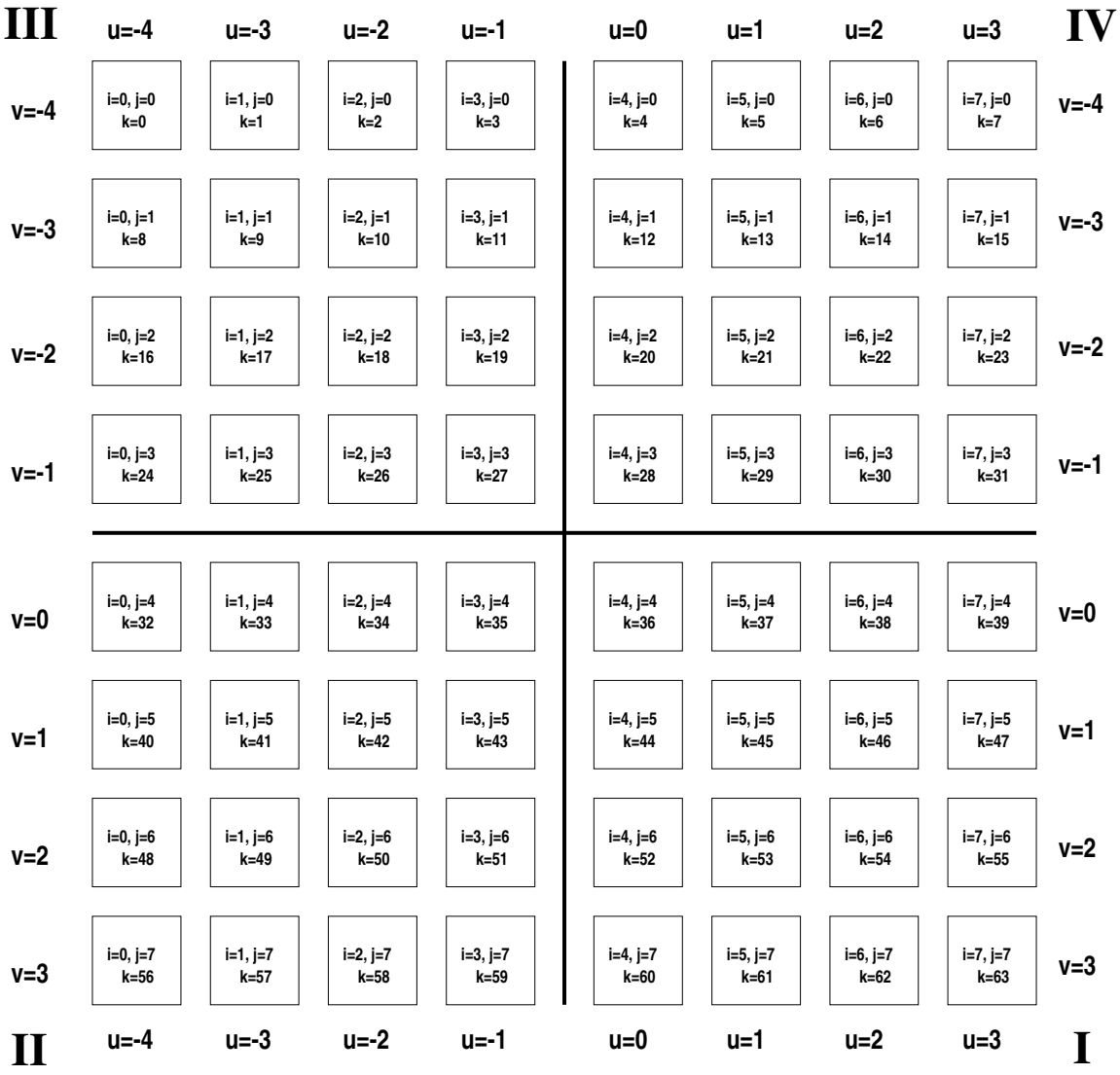
Spring 2007 Dr. Havlicek

- This handout contains a listing of a C program called `DoFFT` that makes a floating point image, computes the DFT of the image, and writes the real and imaginary parts of the DFT out to float files.
- The function that computes the 2D DFT is called `fft2d`. `fft2d` calls a package named FFTW3. To use `fft2d` in one of your programs, you must already have FFTW3 installed.

  You can use the routine `fft2d` in one of three ways:

  1. Include `fft2d` in the same file as your main function. This is how it is done in the sample program `DoFFT`.
  2. Put the code for the `fft2d` routine in a separate file from the main program. This makes the command to compile your program more complicated. The easiest way to handle this is to use a makefile. If you use this way, then be sure to include `fftw3.h` in both the file that contains the `fft2d` function and the file that contains the main function.
  3. Put `fft2d` in a library and then link your program to the library. This is the best way to do it, but it is also the most complicated.
- The program `DoFFT`, including the main function and `fft2d` is written using traditional C. If you want to convert these functions to ANSII C or C++ or Microsoft C, you will have to change the syntax to strongly type the function parameters.
- Most standard FFT routines return the DFT coefficients in a strange and cryptic order. `fft2d` returns the coefficients in a sane order that is easy to understand. In fact, it is the same order as would be used if you called the Matlab routines `fft2` and `fftshift`. The real and imaginary coefficients are returned in separate floating point arrays. The ordering of the coefficients in each of these arrays is shown in the figure on page 2.
- `fft2d` has an integer parameter called `direction`. If you call it with `direction = 1`, then a forward DFT is performed. If you call it with `direction = -1`, a reverse (or inverse) DFT is performed. In either case, the real part of the image is in the array `xReal`, the imaginary part of the image is in the array `xImag`, the real DFT coefficients are in the array `XReal`, and the imaginary DFT coefficients are in the array `XImag`.
- The parameter `size` is the number of rows and columns in the image, which is equal to N on page 4.9 of the notes. `size` must be even for `fft2d` to work and the image must be square.
- On page 4.9 of the notes, there is a factor of $\frac{1}{N}$ in front of both the DFT and the IDFT. `fft2d` does things slightly differently. With `fft2d`, there is a factor of $\frac{1}{N^2}$ in front of the IDFT, but there is no factor in front of the DFT.
- An ascii file containing the source code for DoFFT.c may be obtained on the course web site.
- The results of running the program DoFFT are shown after the listing of the program.

Coefficient Ordering for fft2d:



```
/*
 * DoFFT.c
 *
 *   This program computes the FFT of an image.  Each pixel of the image
 *   is a complex floating point number.  The real part of the image is in
 *   the float array xReal and the imaginary part of the image is in the float
 *   array xImag.
 *
 *   After the 2D FFT routine is called, the real part of the DFT is in the
 *   float array XReal and the imaginary part of the DFT is in the float array
 *   XImag.
 *
 *   - The FFT routine is called fft2d().  This routine operates on
 *     COMPLEX VALUED FLOATING POINT data.  To use it with a real valued
 *     byte per pixel image, you MUST first convert the image to floating point
 *     (set the imaginary part of the floating point image equal to zero).
 *
 *   - fft2d() calls FFTW3 to actually perform the 2D FFT computation.  This
 *     program will not work unless FFTW3 has already been installed.
 *
 *   If you want to look at the real and imaginary parts of the DFT as images,
 *   you must use point operations to convert the floating point arrays into
 *   byte arrays (with full scale contrast).
 *
 *   To look at the magnitude and/or phase of the DFT as images, you must
 *   first compute the magnitude and/or phase as float arrays, and then
 *   convert them to byte arrays (with full scale contrast).  For the
 *   magnitude array, you should add one and perform a logarithmic point
 *   operation before converting to a byte array.
 *
 * OTHER NOTES:
 *
 *   the variable "size" in this program is the same as "N" on page 4.9 of the
 *   notes.
 *
 *   The first thing this program does is make an image that is the sum of
 *   two sinusoidal images.  x1 is a real valued float cosine image with
 *   horizontal frequency u1 = 8 cycles/image and vertical frequency
 *   v1 = 8 cycles/image.  The real part of x1 is in the float array x1Real
 *   and the imaginary part (which is everywhere equal to zero) is in the float
 *   array x1Imag.  x2 is a real valued float sine image with horizontal
 *   frequency u2 = -4 cycles/image and vertical frequency v2 = 7 cycles/image.
 *   The real part of x2 is in the float array x2Real and the imaginary part
 *   (which is everywhere equal to zero) is in the float array x2Imag.
 *
 *   The float image x is equal to the sum of x1 and x2.  The real part of
 *   x is in the float array xReal and the imaginary part is in the float
 *   array xImag.
 *
 *   The program computes the DFT of the image x.  The real part of the DFT
 *   is in the float array XReal and the imaginary part of the DFT is in the
 *   float array XImag.
 *
 * USAGE:
 *
```

```
 *   DoFFT size RealFn ImagFn
 *
 *      size           the image has size rows and size columns.
 *      RealFn         output filename for the real part of the DFT.
 *      ImagFn         output filename for the imaginary part of the DFT.
 *
 * BUILD:
 *
 *   gcc DoFFT.c -o DoFFT -lfftw3 -lm
 *
 * jph 2/18/98
 *
 *   2/28/07:  convert from numerical recipes fourn() to FFTW3 for FFT. jph.
 *
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include "fftw3.h"

/*
 * Function prototypes
 */
void fft2d();

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

main(argc,argv)

  int     argc;
  char   *argv[];
{

  int     size;                 /* row/col dimension of image */

  float  *x1Real;               /* real part of image x1 */
  float  *x1Imag;               /* imaginary part of image x1 */
  float  *x2Real;               /* real part of image x2 */
  float  *x2Imag;               /* imaginary part of image x2 */
  float  *xReal;                /* real part of image x */
  float  *xImag;                /* imaginary part of image x */
  float  *XReal;                /* real part of DFT of image x */
  float  *XImag;                /* imaginary part of DFT of image x */

  double  u1;                   /* horizontal frequency of image x1 */
  double  v1;                   /* vertical frequency of image x1 */
  double  u2;                   /* horizontal frequency of image x2 */
  double  v2;                   /* vertical frequency of image x2 */

  int     row;                  /* image row counter */
```

```
  int     col;                  /* image column counter */
  int     i;                    /* loop counter */

  char   *RealFn;               /* output filename for real part of DFT */
  char   *ImagFn;               /* output filename for imaginary part of DFT */
  int     fd;                   /* file descriptor */
  int     n_bytes;              /* number of bytes to write */

  double  TwoPiOnN;             /* 2pi/size */

/*
 * check for proper invocation and parse arguments
 */
 if (argc != 4) {
   printf("\n%s: Compute the DFT of an image.", argv[0]);
   printf("\nUsage: %s size RealFn ImagFn\n",argv[0]);
   exit(0);
 }
 size = atoi(argv[1]);
 RealFn = argv[2];
 ImagFn = argv[3];

/*
 * Allocate image arrays
 */
 if ((x1Real = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((x1Imag = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((x2Real = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((x2Imag = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((xReal = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((xImag = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((XReal = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((XImag = (float *)malloc(size*size*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
```

```
  exit(-1);
 }

/*
 * Set up a useful constant
 */
 TwoPiOnN = (double)(2.0*M_PI)/(double)size;

/*
 * Make x1 a real cosine image with u1 = v1 = 8 cpi. See page 4.9 of the notes.
 */
 u1 = (double)8.0;
 v1 = (double)8.0;
 for (i=row=0; row < size; row++) {
   for (col=0; col < size; col++,i++) {
     x1Real[i] = (float)cos(TwoPiOnN * (u1*col + v1*row));
     x1Imag[i] = (float)0.0;
   }
 }

/*
 * Make x2 a real sine image with u2 = -4 cpi and v2 = 7 cpi.  See page 4.9.
 */
 u2 = (double)-4.0;
 v2 = (double) 7.0;
 for (i=row=0; row < size; row++) {
   for (col=0; col < size; col++,i++) {
     x2Real[i] = (float)sin(TwoPiOnN * (u2*col + v2*row));
     x2Imag[i] = (float)0.0;
   }
 }

/*
 * Make x = x1 + x2
 */
 for (i=0; i < size*size; i++) {
   xReal[i] = x1Real[i] + x2Real[i];
   xImag[i] = x1Imag[i] + x2Imag[i];
 }

/*
 * Initialize the spectrum of x to zero (this isn't really necessary)
 */
 for (i=0; i < size*size; i++) {
   XReal[i] = XImag[i] = (float)0.0;
 }

/*
 * Compute the DFT of x
 */
 fft2d(xReal,xImag,XReal,XImag,size,1);

/*
 * Write the real and imaginary parts of the DFT out to float files
 */
```

```
 n_bytes = size * size * sizeof(float);
 if ((fd = open(RealFn, O_WRONLY | O_CREAT | O_TRUNC, 0644))==-1) {
   printf("\n%s: could not open file %s!\n\n",argv[0],RealFn);
   exit(-1);
 }
 if (write(fd,XReal,n_bytes) != n_bytes) {
   printf("\n%s: complete disk write of %s did not succeed.",argv[0],RealFn);
 }
 close(fd);

 if ((fd = open(ImagFn, O_WRONLY | O_CREAT | O_TRUNC, 0644))==-1) {
   printf("\n%s: could not open file %s!\n\n",argv[0],ImagFn);
   exit(-1);
 }
 if (write(fd,XImag,n_bytes) != n_bytes) {
   printf("\n%s: complete disk write of %s did not succeed.",argv[0],ImagFn);
 }
 close(fd);

 return;
} /*------------------------  main  ----------------------------------------*/

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

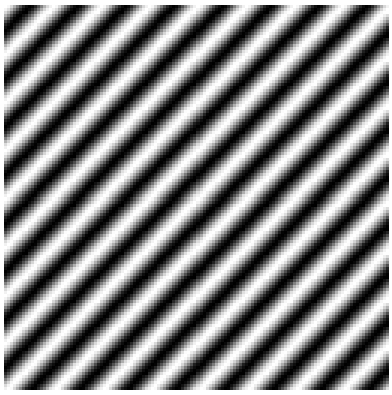


Image $x1$

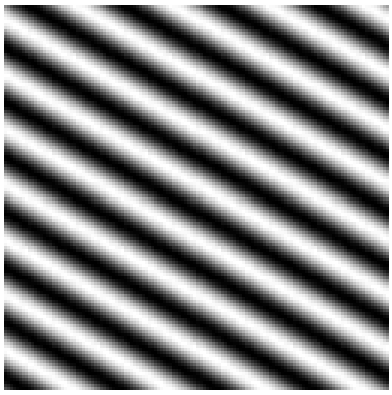


Image $x2$

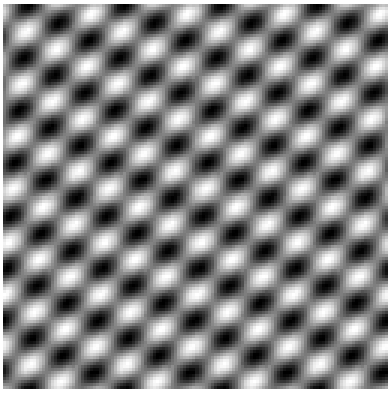
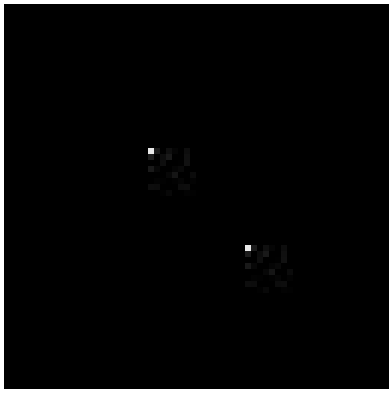
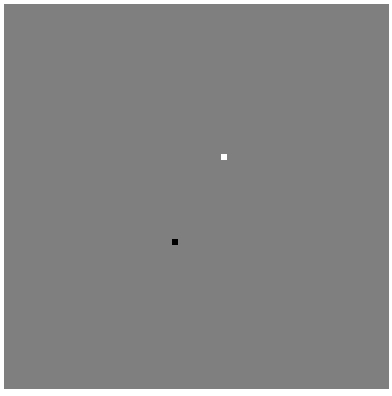


Image $x = x1 + x2$



$\mathrm{Re}\,[\mathrm{DFT}(x)]$



$\mathrm{Im}\,[\mathrm{DFT}(x)]$