ECE 5273
HW 3 Solution

Spring 2023 Dr. Havlicek

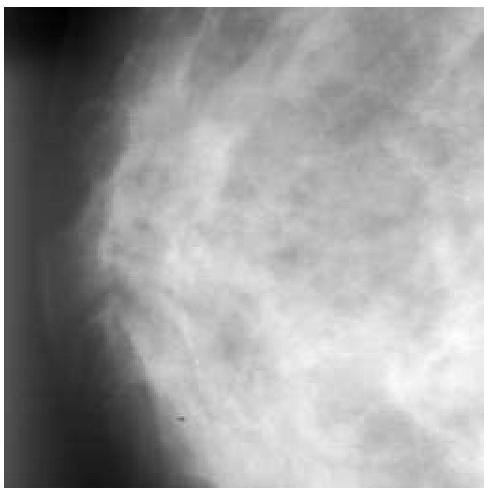
Note: This document contains solutions in both Matlab and traditional C.

C Solution:

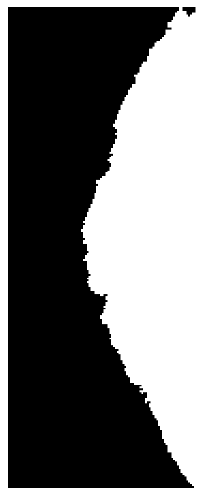
1.

(a):

Mammogram.bin

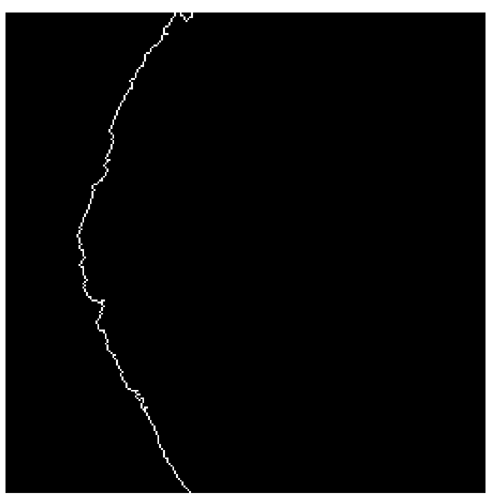


Binary Image; $\tau = 95$



(b):

Approximate Contour Image



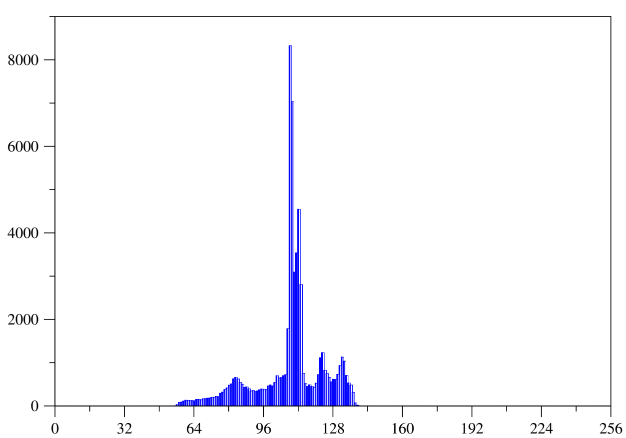
(c) A chain code could not be used to represent the major contour, since the contour generation algorithm is only approximate. Specifically, in a true contour image each logical ONE pixel can have at most two 8-neighbors that are also logical ONE. However, in the contour shown above there are logical ONE pixels that have three or more 8-neighbors that are also logical ONE. The chain code algorithm would fail to determine a unique direction code at such pixels.

2.

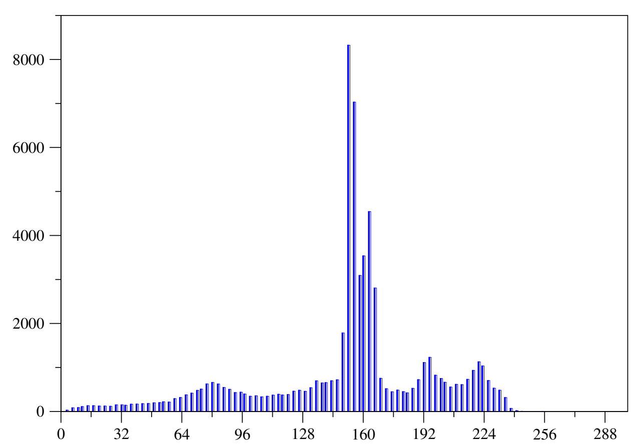
lady.bin

full-scale stretch

Histogram of Original Lady



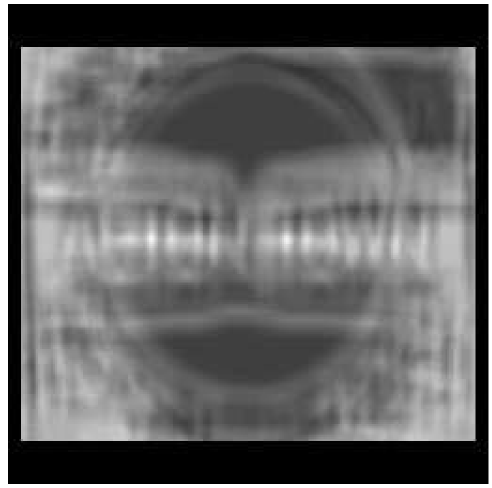
Histogram of Lady After Full-Scale Stretch



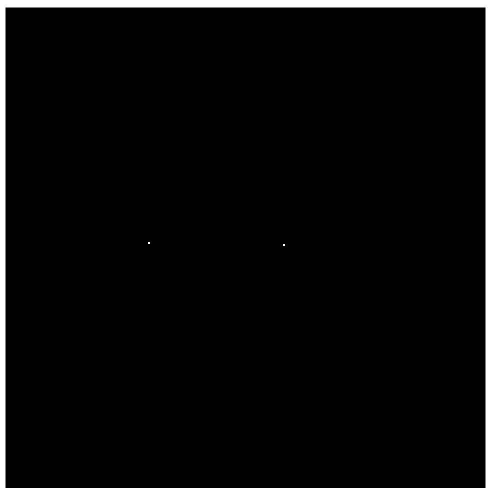
3.

actontBin.bin

$\mathbf{J}_1 = M_2(i,j)$

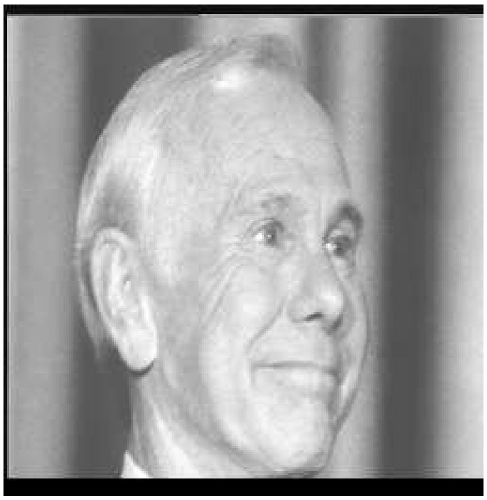


$\mathbf{J}_2$; $\tau = 687$

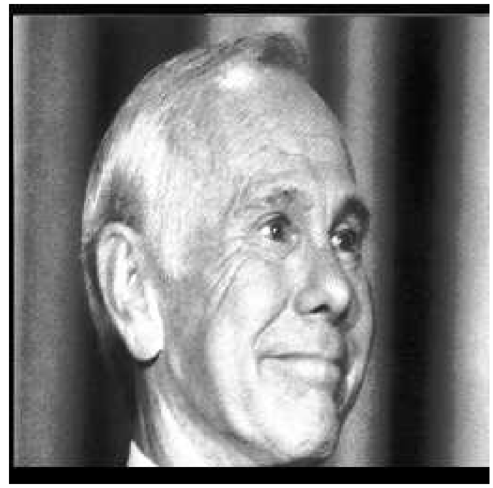


4.

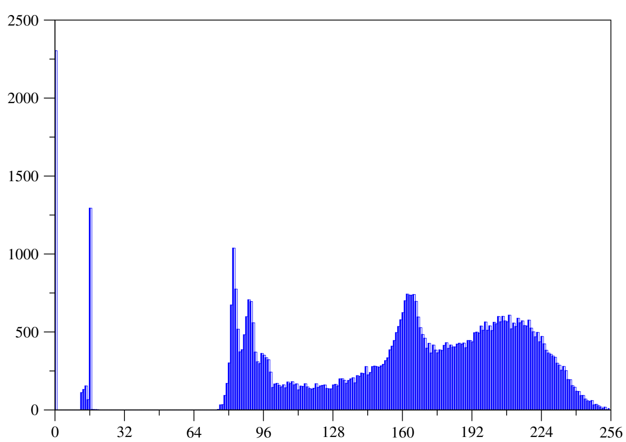
johnny.bin



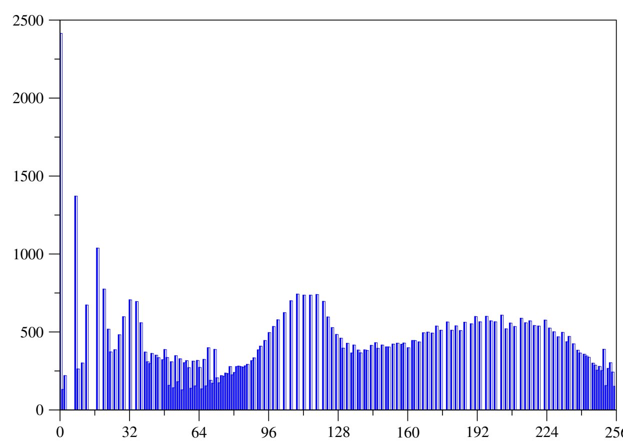
Histogram Equalized Image



Histogram of Original Johnny Image



Histogram of Equalized Johnny Image



C program listings:

```
/*
 * bthresh:
 *
 *   Turn a byte image into a binary image by thresholding at a specified
 *   value.  For pixels that exceed threshold, the output is 255.  Otherwise
 *   the output is 0.  The output uses one byte per pixel.
 *
 *   The input image must be square.
 *
 *
 * 10/3/2000 jph
 *
 */

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

#define BYTE unsigned char

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




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

main(argc,argv)

  int     argc;
  char   *argv[];
{
```

```
  int      i;                    /* counter */
  int      size;                 /* num rows/cols in image */
  BYTE     thresh;               /* threshold value to be used */
  BYTE    *x;                    /* input image */
  BYTE    *y;                    /* output image */
  char    *infile;               /* input filename */
  char    *outfile;              /* output filename */

 /*
  * Check for proper invocation, parse args
  */
  if (argc != 5) {
    printf(
   "\n%s: converts sizeXsize byte image to a binary byte image by thesholding.",
       argv[0]);
    printf("\nUsage: %s size thresh infile outfile\n",argv[0]);
    exit(0);
  }
  size = atoi(argv[1]);
  thresh = (BYTE)atoi(argv[2]);
  infile = argv[3];
  outfile = argv[4];

 /*
  * Allocate image arrays
  */
  if ((x = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((y = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }

 /*
  * Read the input image
  */
  disk2byte(x,size,size,infile);

 /*
  * Threshold the image
  */
  for (i=0; i<size*size; i++) {
    if (x[i] < thresh) {
      y[i] = 0x00;
    }
    else {
      y[i] = 0xff;
    }
  }

 /*
  * Write the output image
  */
  byte2disk(y,size,size,outfile);

  return;
}
```

```
/*
 * ApprCont:
 *
 *  Read a binary image: gray level 0 ==   logical ZERO
 *                       gray level 255 == logical ONE
 *
 *  Output an approximate contour image using algorithm on page 2.105 of notes.
 *
 *
 * 10/3/2000 jph
 * 02/23/2022: update page number refs to the notes. jph
 *
 */

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

#define BYTE unsigned char
#define ONE 0xff
#define ZERO 0x00

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



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

main(argc,argv)

  int     argc;
  char   *argv[];
{
```

```
 int      size;                  /* num rows/cols in image */
 char    *infile;                /* input filename */
 char    *outfile;               /* output filename */
 BYTE    *x;                     /* input image */
 BYTE    *y;                     /* output image */
 int      i;                     /* counter */
 int      row;                   /* image row counter */
 int      col;                   /* image column counter */


/*
 * Check for proper invocation, parse args
 */
 if (argc != 4) {
   printf(
  "\n%s: approximate contour image generation.", argv[0]);
   printf("\nUsage: %s size infile outfile\n",argv[0]);
   exit(0);
 }
 size = atoi(argv[1]);
 infile = argv[2];
 outfile = argv[3];

/*
 * Allocate image arrays
 */
 if ((x = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((y = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }

/*
 * Read the input image
 */
 disk2byte(x,size,size,infile);

/*
 * Make output image using the algorithm on page 2.105 of the notes
 */
 for (i=row=0; row < size; row++) {
   for (col=0; col < size; col++,i++) {
     if (x[i] == ONE) {                 /* input pixel is logical ONE */
       y[i] = ZERO;
     } else {                           /* input pixel is logical ZERO */
       if (
         ((col > 0) && (x[i-1] == ONE))  /* left neighbor is logical ONE */
       ||((col < size) && (x[i+1] == ONE)) /* right neighbor is logical ONE */
       ||((row > 0) && (x[i-size] == ONE))  /* upper neighbor is logical ONE */
       ||((row < size) && (x[i+size] == ONE))/* lower neighbor is logical ONE*/
       ) {
         y[i] = ONE;
       } else {
         y[i] = ZERO;
       }             /* else */
     }               /* else */
   }                 /* for col */
 }                   /* for row */

/*
 * Write the output image
 */
 byte2disk(y,size,size,outfile);

 return;
```

```
}

/*
 * CStretch:
 *
 *   - Read in a byte image.
 *   - Write the histogram to an ascii file.
 *   - Perform a full scale contrast stretch.
 *   - Write the new histogram to an ascii file.
 *   - Write the output image.
 *
 *   The input image must be square.
 *
 * 10/3/2000 jph
 *
 */

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

#define BYTE unsigned char

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




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

main(argc,argv)

  int     argc;
  char   *argv[];
```

```
{

  int      size;                  /* num rows/cols in image */
  int      row;                   /* image row counter */
  int      col;                   /* image col counter */
  int      i;                     /* counter */
  char    *infile;                /* input filename */
  char    *outfile;               /* output filename */
  char    *HistInFile;            /* input histogram filename */
  char    *HistOutFile;           /* output histogram filename */
  BYTE    *x;                     /* input image */
  BYTE    *y;                     /* output image */
  BYTE     min;                   /* minimum pixel value in input image */
  BYTE     max;                   /* maximum pixel value in input image */
  float    K;                     /* scale factor for full scale contrast */
  int      hist1[256];            /* histogram of original image */
  int      hist2[256];            /* histogram of output image */
  FILE    *outfp;                 /* ascii output file pointer */

 /*
  * Check for proper invocation, parse args
  */
  if (argc != 6) {
    printf(
   "\n%s: byte image full scale contrast stretch, output ascii histograms.",
       argv[0]);
    printf("\nUsage: %s size infile outfile HistInFile HistOutFile\n",argv[0]);
    exit(0);
  }
  size = atoi(argv[1]);
  infile = argv[2];
  outfile = argv[3];
  HistInFile = argv[4];
  HistOutFile = argv[5];

 /*
  * Allocate image arrays
  */
  if ((x = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }
  if ((y = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
    printf("\n%s: free store exhausted.\n",argv[0]);
    exit(-1);
  }

 /*
  * Initialize histograms
  */
  for (i=0; i < 256; i++) {
    hist1[i] = hist2[i] = 0;
  }

 /*
  * Read the input image
  */
  disk2byte(x,size,size,infile);

 /*
  * Full scale contrast stretch
  */
  min = max = x[0];
  for (i=0; i < size*size; i++) {
    if (x[i] < min) {
      min = x[i];
    } else {
      if (x[i] > max) {
```

```
        max = x[i];
      }
    }
  }
  K = (float)255.0 / (float)(max-min);
  for (i=0; i < size*size; i++) {
    y[i] = (BYTE)((float)(x[i] - min)*K + 0.5);
  }

 /*
  * Compute histograms
  */
  for (i=0; i < size*size; i++) {
    hist1[x[i]]++;
    hist2[y[i]]++;
  }

 /*
  * Write the output image
  */
  byte2disk(y,size,size,outfile);

 /*
  * Write input histogram
  */
  if ((outfp = fopen(HistInFile,"w")) == NULL) {
    printf("\nCannot open file %s",HistInFile);
    exit(0);
  }
  for (i=0; i<256; i++) {
    fprintf(outfp,"%d\t%d\n",i,hist1[i]);
  }
  fclose(outfp);

 /*
  * Write output histogram
  */
  if ((outfp = fopen(HistOutFile,"w")) == NULL) {
    printf("\nCannot open file %s",HistOutFile);
    exit(0);
  }
  for (i=0; i<256; i++) {
    fprintf(outfp,"%d\t%d\n",i,hist2[i]);
  }
  fclose(outfp);


  return;
}
```

```
/*
 * FindT:
 *
 *  Read a binary image: gray level 0 ==   logical ZERO
 *                       gray level 255 == logical ONE
 *
 *  Find occurrences of the letter "T" based on template made from the image
 *  actontBin.bin.
 *
 *
 * 10/3/2000 jph
 *  2/8/2015: thresholding now done directly on J1 and in this file.
 *
 */

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

#define BYTE unsigned char
#define ONE 0xff
#define ZERO 0x00

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



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

main(argc,argv)

  int     argc;
  char   *argv[];
{
```

```
 int      size;                 /* num rows/cols in image */
 char    *infile;               /* input filename */
 char    *J1outfile;            /* output filename for J1 */
 char    *J2outfile;            /* output filename for J2 */
 BYTE    *x;                    /* input image */
 BYTE    *y;                    /* output J1 image with full-scale stretch */
 BYTE    *Templ;                /* template */
 int     *J1;                   /* image containing M2 match measure */
 BYTE    *J2;                   /* Thresholded J1 image: location of T's */
 int      Big1;                 /* biggest count in J1 image */
 int      Big2;                 /* 2nd biggest count in J1 image */
 int      i;                    /* counter */
 int      row;                  /* image row counter */
 int      col;                  /* image column counter */
 int      TemplRows;            /* number of rows in template */
 int      TemplRowso2;          /* TemplRows / 2 */
 int      TemplCols;            /* number of columns in template */
 int      TemplColso2;          /* TemplCols / 2 */
 int      TRow;                 /* current row in template */
 int      TCol;                 /* current column in template */
 int      XRow;                 /* current row in image */
 int      XCol;                 /* current column in image */
 float    min;                  /* minimum value of M2 match measure in J1 */
 float    max;                  /* maximum value of M2 match measure in J1 */
 float    K;                    /* scale factor for full scale contrast */



/*
 * Check for proper invocation, parse args
 */
 if (argc != 5) {
   printf(
  "\n%s: template matching for the letter T.", argv[0]);
   printf("\nUsage: %s size infile J1outfile J2outfile\n",argv[0]);
   exit(0);
 }
 size = atoi(argv[1]);
 infile = argv[2];
 J1outfile = argv[3];
 J2outfile = argv[4];

/*
 * Allocate image arrays
 */
 if ((x = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((y = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((J1 = (int *)malloc(size*size*sizeof(int))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((J2 = (BYTE *)malloc(size*size*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }

/*
 * Read the input image
 */
 disk2byte(x,size,size,infile);
```

```
/*
 * Construct template
 */
 TemplRows = 47;
 TemplRowso2 = TemplRows >> 1;
 TemplCols = 15;
 TemplColso2 = TemplCols >> 1;

 if ((Templ = (BYTE *)malloc(TemplRows*TemplCols*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }

 for (row=0; row < 10; row++) {
   for (col=0; col < TemplCols; col++) {
     Templ[row*TemplCols + col] = ZERO;
   }
 }
 for ( ; row < 16; row++) {
   for (col=0; col < TemplCols; col++) {
     Templ[row*TemplCols + col] = ONE;
   }
 }
 for ( ; row < 37; row++) {
   for (col=0; col < 6; col++) {
     Templ[row*TemplCols + col] = ZERO;
   }
   for ( ; col < 10; col++) {
     Templ[row*TemplCols + col] = ONE;
   }
   for ( ; col < TemplCols; col++) {
     Templ[row*TemplCols + col] = ZERO;
   }
 }
 for ( ; row < TemplRows; row++) {
   for (col=0; col < TemplCols; col++) {
     Templ[row*TemplCols + col] = ZERO;
   }
 }

/*
 * Initialize output image J1
 */
 for (i=0; i < size*size; i++) {
   J1[i] = (int)0.0;
 }

/*
 * Loop over image pixels computing the match measure M2
 */
 for (row=TemplRowso2; row < size-TemplRowso2; row++) {
   for (col=TemplColso2; col < size-TemplColso2; col++) {
     for (TRow=0,XRow=row-TemplRowso2; TRow < TemplRows; TRow++,XRow++) {
       for (TCol=0,XCol=col-TemplColso2; TCol < TemplCols; TCol++,XCol++) {
         if (x[XRow*size + XCol] == Templ[TRow*TemplCols + TCol]) {
           J1[row*size + col]++;
         }
       }
     }
   }
 }

/*
 * Convert J1 to output byte image y with full scale contrast
 */
 min = max = (float)J1[0];
 for (i=0; i < size*size; i++) {
   if ((float)J1[i] < min) {
```

```
      min = (float)J1[i];
    } else {
      if ((float)J1[i] > max) {
        max = (float)J1[i];
      }
    }
  }
  K = (float)255.0 / (max - min);
  for (i=0; i < size*size; i++) {
    y[i] = (BYTE)(K*((float)J1[i] - min) + 0.5);
  }

 /*
  * Write the J1 image to a BYTE image file
  */
  byte2disk(y,size,size,J1outfile);

 /*
  * Find top two counts in J1.  2nd largest is threshold.
  */
  Big1 = Big2 = 0;
  for (i=0; i < size*size; i++) {
    if (J1[i] > Big1) {
      Big1 = J1[i];
    } else {
      if (J1[i] > Big2) {
        Big2 = J1[i];
      }
    }
  }

  printf("\nBig2 = %d\n",Big2);

 /*
  * Make J2 image by thresholding J1.  Write J2 to disk.
  */
  for (i=0; i < size*size; i++) {
    if (J1[i] >= Big2) {
      J2[i] = ONE;
    } else {
      J2[i] = ZERO;
    }
  }
  byte2disk(J2,size,size,J2outfile);

  return;
}
```

```
/*
 * bEqualize.c
 *
 *   Perform histogram equalization (flattening) on a byte image using the
 *   algorithm given on pages 3.36 - 3.37 of the course notes.
 *
 * jph 4/2/03
 *
 * 2/12/14: the full-scale contrast stretch part of the algorithm is updated to
 *          agree with the new version of the notes.  jph.
 *
 */

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

#define BYTE unsigned char

/*
 * Function Prototypes (forward declarations)
 */
void disk2byte();
void byte2disk();


main(argc,argv)

  int     argc;
  char   *argv[];
{

  int      size;                  /* num rows/cols in image */
  int      n;                     /* num pixels in image = size*size */
  float    one_on_n;              /* 1/n */
  char    *InFile;                /* input filename (byte image) */
  char    *OutFile;               /* output filename (for equalized image) */
  char    *OrigHistFile;          /* output fn for original histogram */
  char    *EqHistFile;            /* output fn for equalized histogram */
  BYTE    *x;                     /* input image */
  float   *J;                     /* intermediate image J */
  float    maxJ;                  /* max pixel value in J image */
  float    minJ;                  /* min pixel value in J image */
  BYTE    *K;                     /* output image (equalized) */
  float   *OrigHist;              /* histogram of original image */
  float   *p;                     /* normalized histogram of original image */
  float   *P;                     /* cumulative histogram of original image */
  float   *EqHist;                /* histogram of equalized image */
  int      i;                     /* counter */
  int      k;                     /* counter */
  FILE    *pAsciiFile;            /* file ptr for ascii output files */

 /*
  * Check for proper invocation, parse args
  */
  if (argc != 6) {
    printf(
   "\n%s: perform histogram equalization on a BYTE image.", argv[0]);
    printf("\nUsage: %s size infile outfile OrigHistFile EqHistFile\n",
           argv[0]);
    exit(0);
  }
  size = atoi(argv[1]);
  InFile = argv[2];
  OutFile = argv[3];
  OrigHistFile = argv[4];
```

```
 EqHistFile = argv[5];
 n = size * size;
 one_on_n = (float)1.0 / (float)n;

/*
 * Allocate arrays for images and histograms
 */
 if ((x = (BYTE *)malloc(n*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((J = (float *)malloc(n*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((K = (BYTE *)malloc(n*sizeof(BYTE))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((OrigHist = (float *)malloc(256*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((p = (float *)malloc(256*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((P = (float *)malloc(256*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }
 if ((EqHist = (float *)malloc(256*sizeof(float))) == NULL) {
   printf("\n%s: free store exhausted.\n",argv[0]);
   exit(-1);
 }

/*
 * Read the input image
 */
 disk2byte(x,size,size,InFile);

/*
 * Initialize histograms
 */
 for (i=0; i < 256; i++) {
   OrigHist[i] = p[i] = EqHist[i] = (float)0.0;
 }

/*
 * Compute histogram of original image
 */
 for (i=0; i < n; i++) {
   OrigHist[x[i]]++;
 }

/*
 * Compute the normalized histogram of the original image
 */
 for (i=0; i < 256; i++) {
   p[i] = one_on_n * OrigHist[i];
 }

/*
 * Compute the cumulative histogram of the original image
 */
 P[0] = p[0];
 for (i=1; i < 256; i++) {
   P[i] = P[i-1] + p[i];
```

```
  }

 /*
  * Compute the image J defined on page 3.36 of the notes
  */
  for (i=0; i < n; i++) {
    J[i] = P[x[i]];
  }

  /* find the min and max of J */
  maxJ = -1;  // smaller than any pixel in J
  minJ = 2;   // bigger than any pixel in J
  for (i=0; i < n; i++) {
    if (J[i] > maxJ) {
      maxJ = J[i];
    } else {
      if (J[i] < minJ) {
        minJ = J[i];
      }
    }
  }

  /* do the full-scale contrast stretch (notes p. 3.15) */
  for (i=0; i < n; i++) {
    K[i] = (BYTE)truncf(
             (float)255.0 / ((float)(maxJ - minJ)) * (J[i] - (float)minJ)
             + (float)0.5);
  }

 /*
  * Compute the histogram of the output (equalized) image
  */
  for (i=0; i < n; i++) {
    EqHist[K[i]]++;
  }

 /*
  * Write the output image
  */
  byte2disk(K,size,size,OutFile);

 /*
  * Write the original image histogram out to an ascii file
  */
  if ((pAsciiFile = fopen(OrigHistFile,"w")) == NULL) {
    printf("\n%s: cannot open file %s.\n",argv[0],OrigHistFile);
    exit(-1);
  }
  for (i=0; i < 256; i++) {
    fprintf(pAsciiFile,"%d\t%f\n",i,OrigHist[i]);
  }
  fclose(pAsciiFile);

 /*
  * Write the equalized image histogram out to an ascii file
  */
  if ((pAsciiFile = fopen(EqHistFile,"w")) == NULL) {
    printf("\n%s: cannot open file %s.\n",argv[0],EqHistFile);
    exit(-1);
  }
  for (i=0; i < 256; i++) {
    fprintf(pAsciiFile,"%d\t%f\n",i,EqHist[i]);
  }
  fclose(pAsciiFile);


  return;
} /*----------------  Main  ----------------------------------------------*/
```

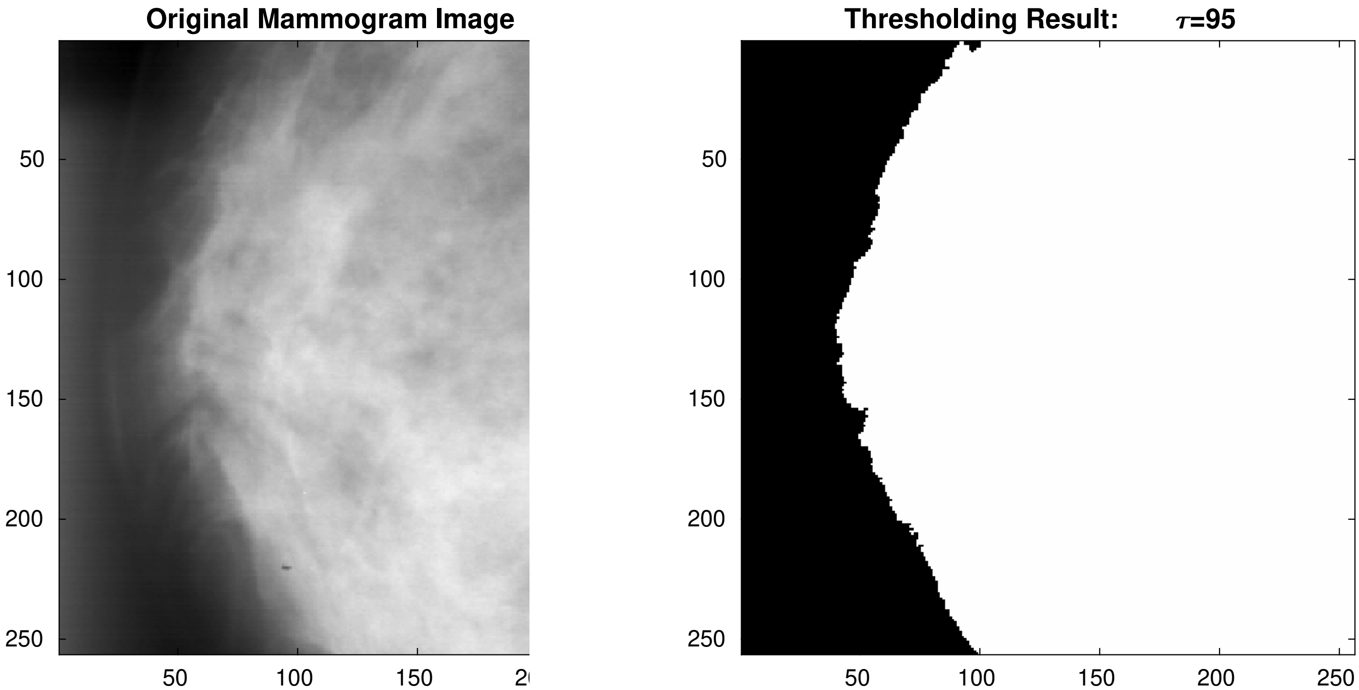
```
/*--------------------------------------------------------------------------
 * disk2byte.c
 *
 *   function reads an unsigned char (byte) image from disk
 *
 *
 *  jph 15 June 1992
 *    12 April 2002: revision for improved error handling, jph
 *
 ---------------------------------------------------------------------------*/

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)) {
    printf("\ndisk2byte.c : row_dim=%d, col_dim=%d !\n\n",row_dim,col_dim);
    exit(-1);
  }

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

 /*
  * 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.\n\n",fn);
    exit(-1);
  }

 /*
  * 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;
}
```

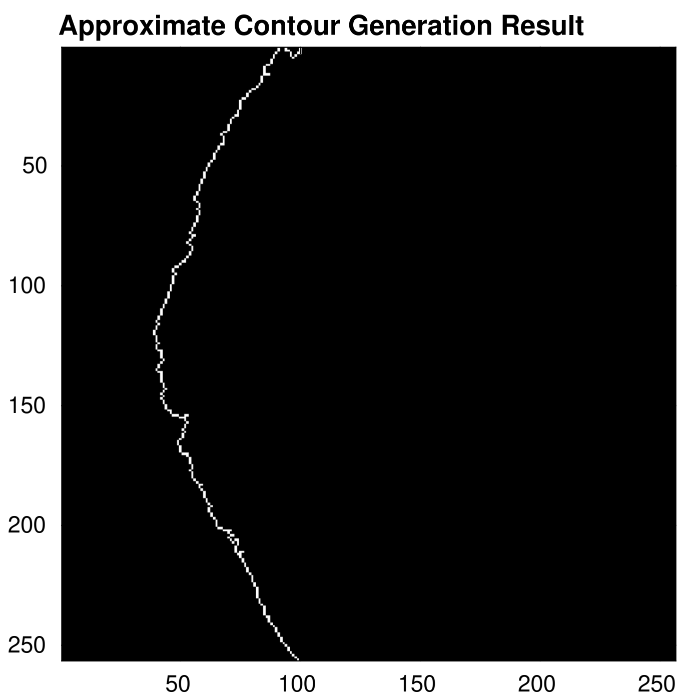
Matlab Solution:

1.

(a):

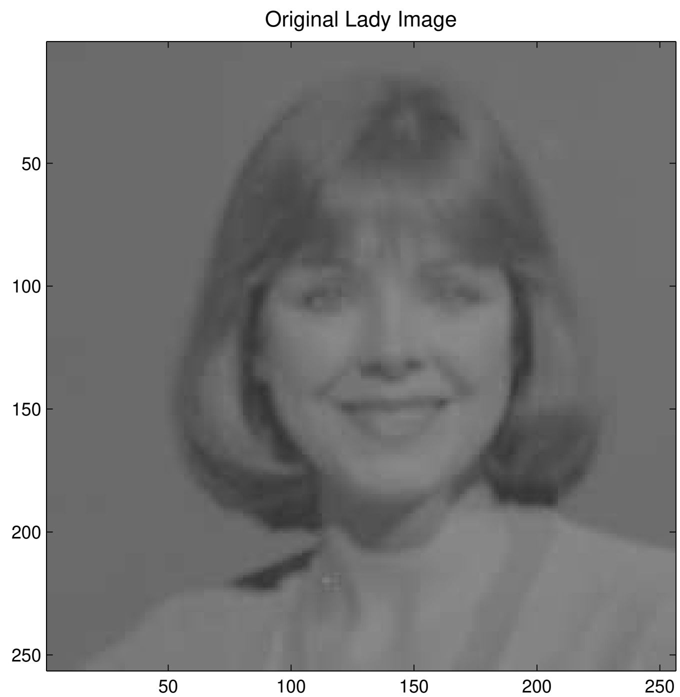


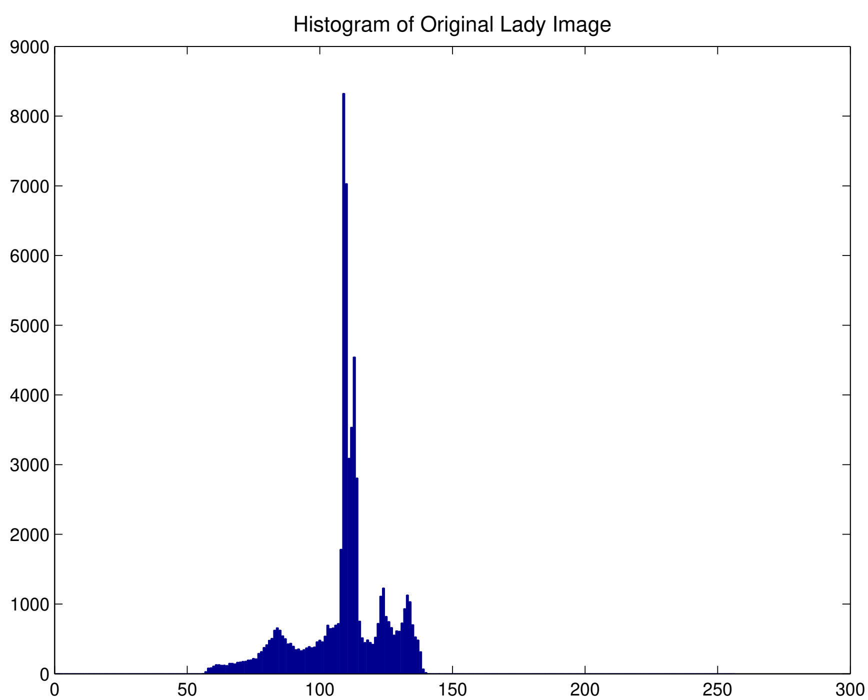
(b):

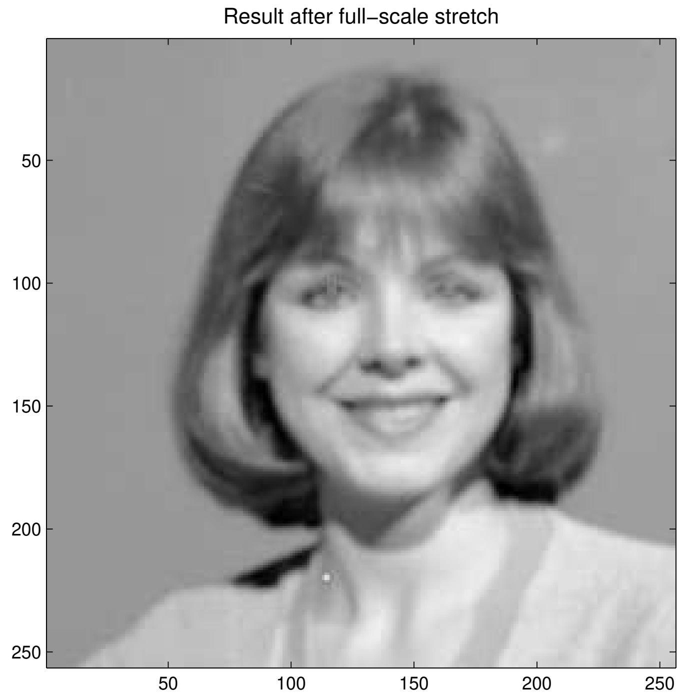


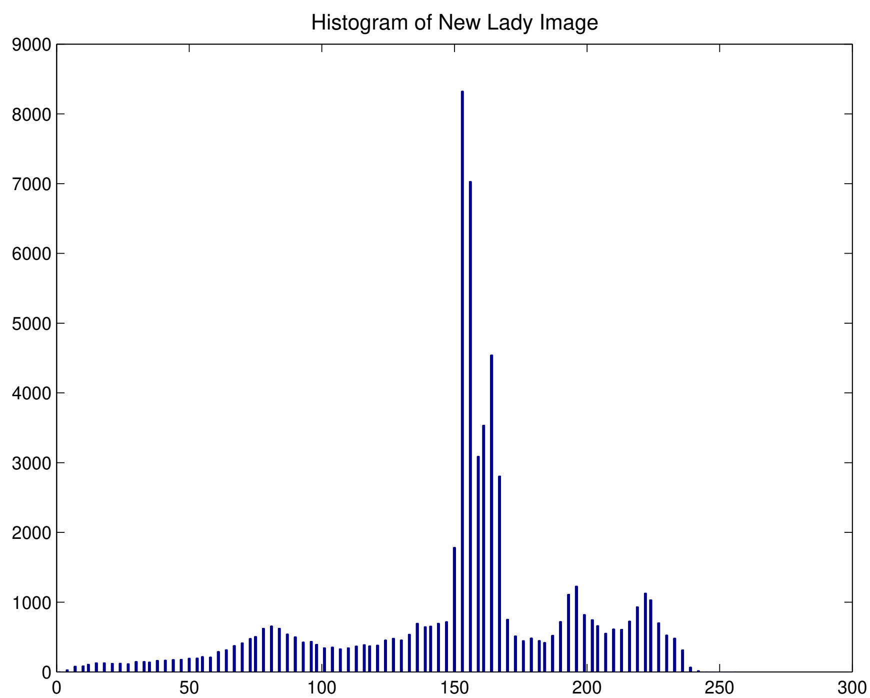
(c) A chain code could not be used to represent the major contour, since the contour generation algorithm is only approximate. Specifically, in a true contour image each logical ONE pixel can have at most two 8-neighbors that are also logical ONE. However, in the contour shown above there are logical ONE pixels that have three or more 8-neighbors that are also logical ONE. The chain code algorithm would fail to determine a unique direction code at such pixels.

2.



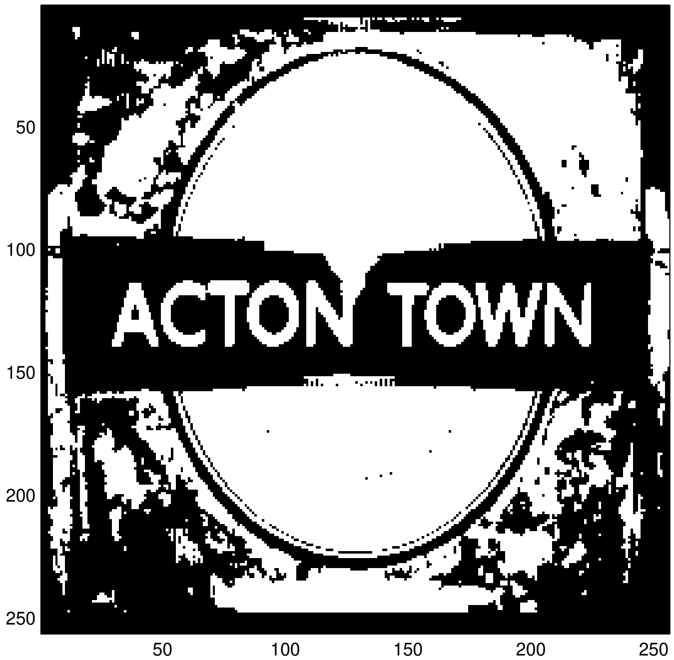



Result after full−scale stretch
50
100
150
200
250
50
100
150
200
250

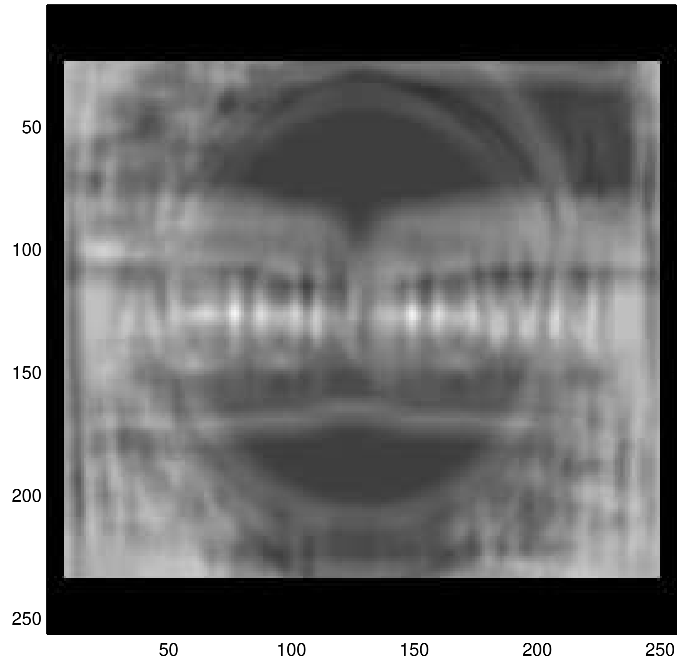

Histogram of New Lady Image
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250
300

3.

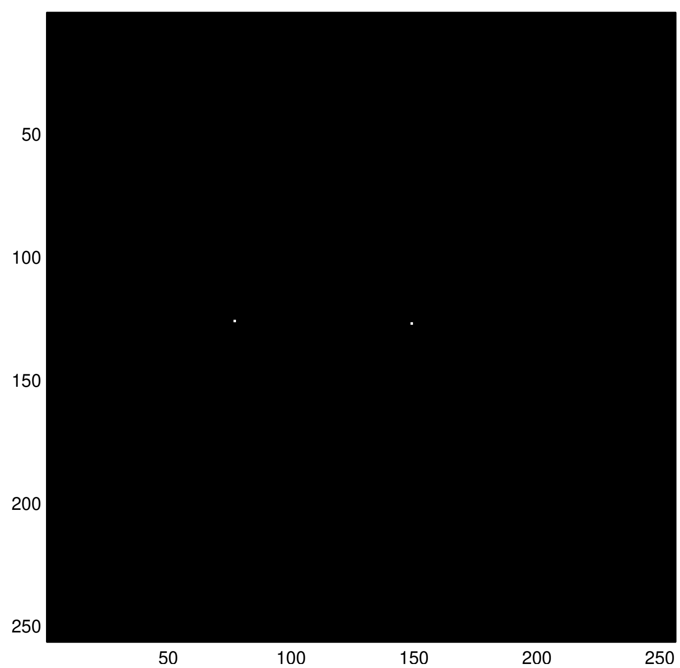
Original actontBin Image



$J1 = M_2(i,j)$

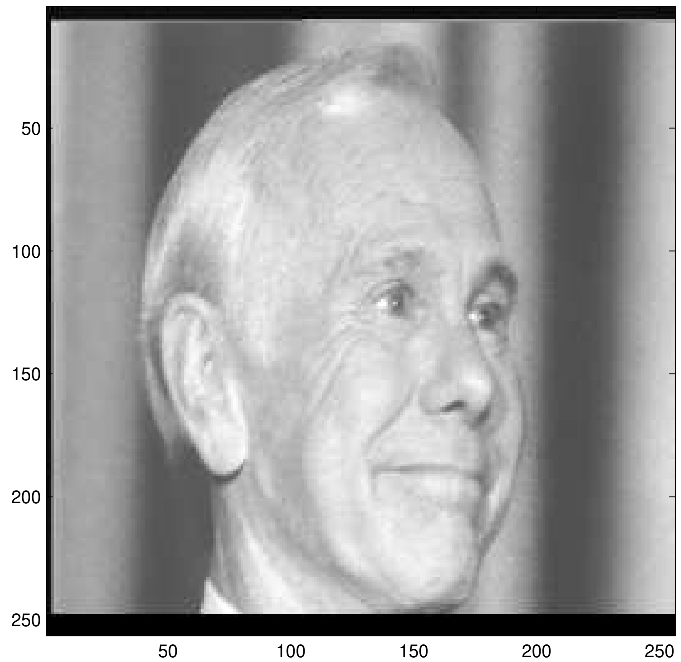


J2; τ=687

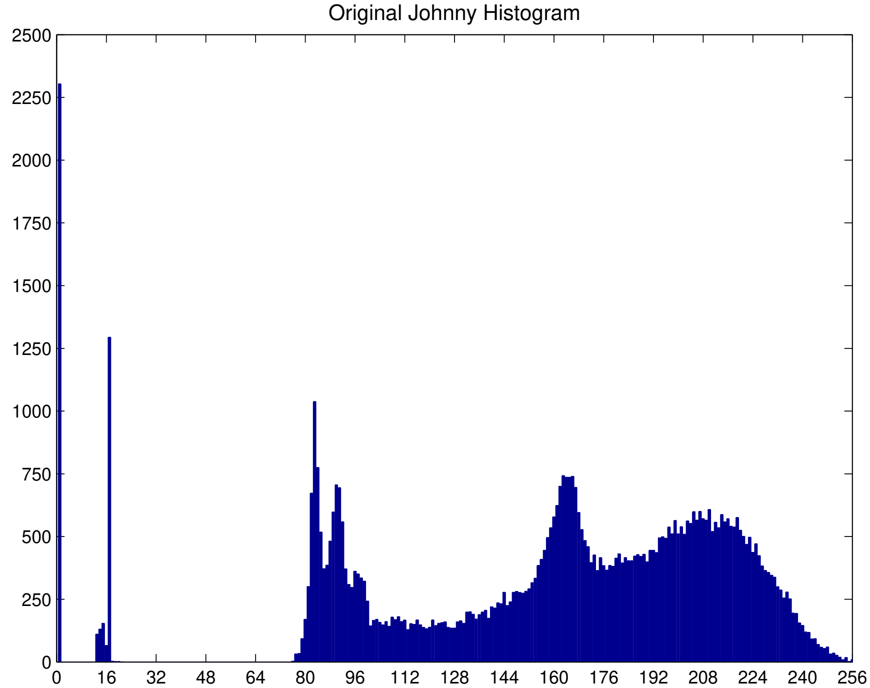


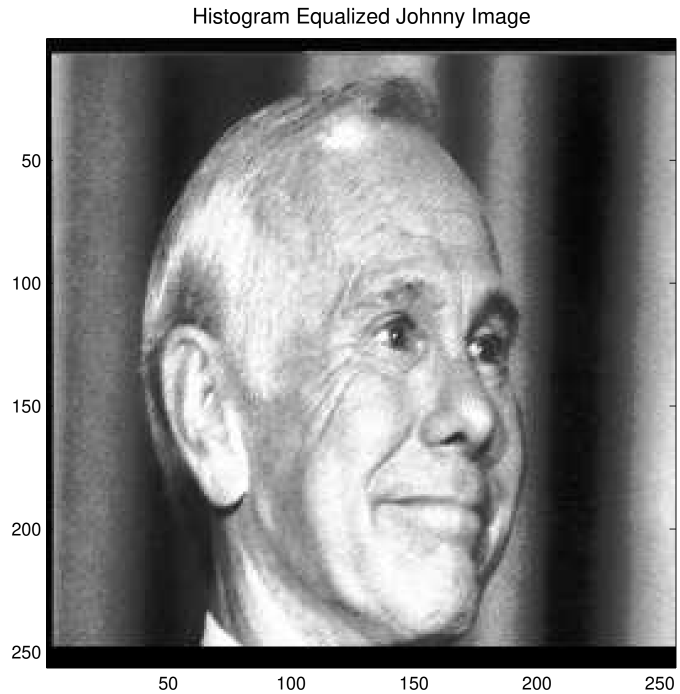
4.

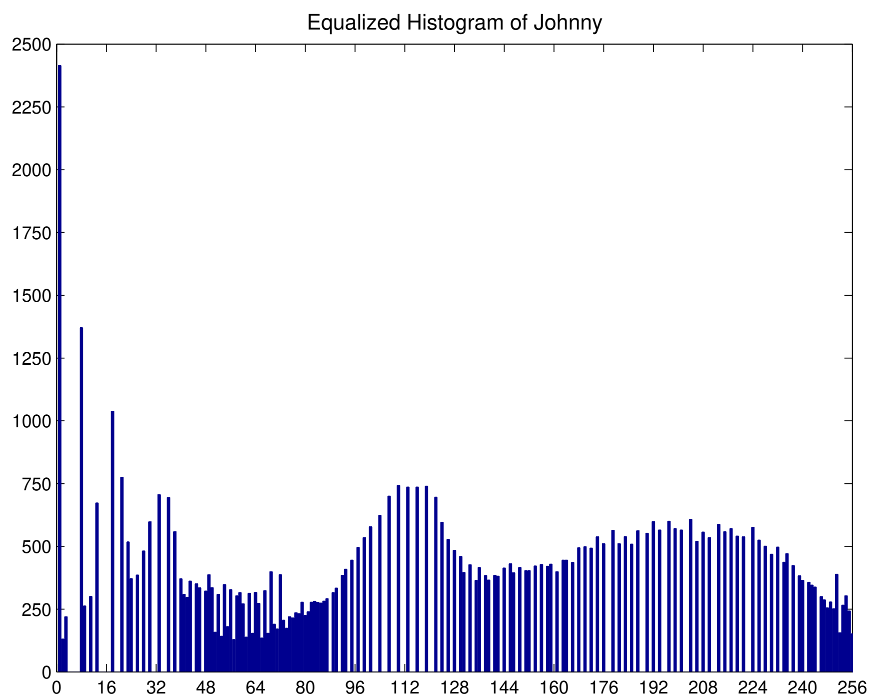
Original johnny.bin Image



Original Johnny Histogram




Histogram Equalized Johnny Image
50
100
150
200
250
50
100
150
200
250


Equalized Histogram of Johnny
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
0
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
256

Listing of Matlab m-files:

```
% Mammogram.m
%
% Mammogram.m
%
%  This is homework 3, problem 1, ECE 5273, Spring 2011.
%
%  jph 21 April 2002
%  jph 21 February 2011: this is now HW 3.
%  jph 04 February 2014: removed loop for threshold to make it run faster.
%  jph 08 February 2015: changed the figure titles
%
%

xsize = 256;

%
% read Mammogram.bin, transpose, display, and print
%
xorig = ReadBin('Mammogram.bin',xsize);
figure(1);colormap(gray(256));
image(xorig);
axis('image');
title('Original Mammogram Image','FontSize',12);
print -deps MMammogramCrop.eps;
print -dpdf MMammogram.pdf;

%
% Threshold at the empirically determined value thresh=95
%
thresh = 95;
x = 255 * (xorig >= thresh);

%
% Display and print thresholding result
%
figure(2);colormap(gray(256));
image(x);axis('image');
title('Thresholding Result: \tau=95','FontSize',12);
print -deps Mresult1Crop.eps;
print -dpdf Mresult1.pdf;

%
% Make output image y using the algorithm on page 2.105 of the notes
%
y = zeros(xsize,xsize);
for row=1:xsize
  for col=1:xsize
    if (x(row,col) == 255)
      y(row,col) = 0;
    else
      if (  ((col > 1) & (x(row,col-1)==255))...    % left  neighbor logic ONE
          | ((col < xsize) & (x(row,col+1)==255))... % right neighbor logic ONE
          | ((row > 1) & (x(row-1,col)==255))...    % upper neighbor logic ONE
          | ((row < xsize) & (x(row+1,col)==255))... % lower neighbor logic ONE
         )
        y(row,col) = 255;
      else
        y(row,col) = 0;
      end
    end
  end
end

%
% Display and print result
%
```

```
figure(3);colormap(gray(256));
image(y);axis('image');
title('Approximate Contour Generation Result','FontSize',12);
print -deps Mresult2Crop.eps;
print -dpdf Mresult2.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';

%
% Lady.m
%
%  This is homework 3, problem 2, ECE 5273, Spring 2015.
%
%  jph 21 April 2002
%  jph 21 February 2005: update histogram call; use 'hist'.
%  jph 21 February 2011: this is now HW 3.
%  jph  8 February 2015: Call subroutines to read image and stretch.
%

xsize = 256;

%
% read lady.bin, transpose, display, and print
%
x = ReadBin('lady.bin',xsize);
figure(1);colormap(gray(256));
image(x);axis('image');
```

```
title('Original Lady Image','FontSize',12);
print -deps MLadyCrop.eps;
print -dpdf MLady.pdf;

%
% Compute, plot, and print histogram of original image
%
histx = sum(hist(x,0:255)');
figure(2);
bar(histx);
title('Histogram of Original Lady Image','FontSize',12);
print -deps MOrigHistCrop.eps;
print -dpdf MOrigHist.pdf;

%
% Perform full-scale contrast stretch, display and print
%
y = stretch(x);
figure(3);colormap(gray(256));
image(y);axis('image');
title('Result after full-scale stretch','FontSize',12);
print -deps MNewLadyCrop.eps
print -dpdf MNewLady.pdf

%
% Compute new histogram, plot and print.
%
histy = sum(hist(y,0:255)');
figure(4);
bar(histy);
title('Histogram of New Lady Image','FontSize',12);
print -deps MFinalHistCrop.eps
print -dpdf MFinalHist.pdf


%
% 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);
```

```
%
% FindT.m
%
%  This is homework 3, problem 3, ECE 5273, Spring 2015.
%
%  jph 21 April 2002
%  jph 21 February 2011: this is now HW 3.
%  jph 04 February 2014: remove some loops to speed it up.
%  jph 08 February 2015: compute threshold automatically,
%                        threshold J2 directly instead of y
%

xsize = 256;

%
% read actontBin.bin, transpose, display, and print
%
x = ReadBin('actontBin.bin',xsize);
figure(1);colormap(gray(256));
image(x);axis('image');
title('Original actontBin Image','FontSize',12);
print -deps MactontBinCrop.eps;
print -dpdf MactontBin.pdf;

%
% Construct template based on analysis of the image
%
TemplRows = 47;
TemplRowso2 = floor(TemplRows/2);
TemplCols = 15;
TemplColso2 = floor(TemplCols/2);

Templ = zeros(TemplRows,TemplCols);

for row=1:10
  for col=1:TemplCols
    Templ(row,col) = 0;
  end
end
for row=11:16
  for col=1:TemplCols
    Templ(row,col) = 255;
  end
end
for row=17:37
  for col=1:6
    Templ(row,col) = 0;
  end
  for col=7:10
    Templ(row,col) = 255;
  end
  for col=11:TemplCols
    Templ(row,col) = 0;
  end
end
for row=38:TemplRows
  for col=1:TemplCols
    Templ(row,col) = 0;
  end
end

%
% Initialize output image J1
%
J1 = zeros(xsize,xsize);

%
% Loop over image pixels computing the match measure M2
```

```
%
for row=TemplRowso2+1:xsize-TemplRowso2
  for col=TemplColso2+1:xsize-TemplColso2
    WindowSet = ...
           x(row-TemplRowso2:row+TemplRowso2,col-TemplColso2:col+TemplColso2);
    J1(row,col) = sum(sum(WindowSet == Templ));
  end
end

%
% Perform full-scale stretch.  Display and write J1.
%
y = stretch(J1);
figure(2);colormap(gray(256));
image(y);axis('image');
title('J1 = M_2(i,j)','FontSize',12);
print -deps MJ1Crop.eps;
print -dpdf MJ1.pdf;

%
% Create J2
%
% The two highest counts M2(i,j) in J1 should correspond to the centers of
% the two T's.  Use this fact to compute the threshold.
%
counts = sort(J1(:),'descend');
thresh = counts(2);
J2 = 255 * (J1 >= thresh);

%
% display and write J2
%
figure(3);colormap(gray(256));
image(J2);axis('image');
title(sprintf('J2; \\tau=%d',thresh),'FontSize',12);
print -deps MJ2Crop.eps;
print -dpdf MJ2.pdf;

%
% JohnnyEQ.m
%
% 04/03/03 jph
%
% 02/12/14: update full-scale contrast stretch part of histogram
%           equalization algorithm to match new notes. jph.
%


xsize = 256;


%
% Read and display the image
%
x = ReadBin('johnny.bin',xsize);
figure(1);colormap(gray(256));
image(x);axis('image');
title('Original johnny.bin Image','FontSize',12);
print -deps MjohnnyCrop.eps;
print -dpdf Mjohnny.pdf;

%
% Show the histogram of the original image
%
figure(2);
bar(sum(hist(x,0:255)'));
title('Original Johnny Histogram','FontSize',12);
axis([0 256 0 2500]);
```

```
set(gca,'XTick',[0:16:256]);
set(gca,'YTick',[0:250:2500]);
print -deps MjohnnyOrigHistCrop.eps;
print -dpdf MjohnnyOrigHist.pdf;

%
% Perform histogram equalization
%
y = HistEQ(x);

%
% Display the new image and show its histogram
%
figure(3);colormap(gray(256));
image(y);axis('image');
title('Histogram Equalized Johnny Image','FontSize',12);
print -deps MjohnnyEQCrop.eps;
print -dpdf MjohnnyEQ.pdf;

figure(4);
bar(sum(hist(y,0:255)'));
title('Equalized Histogram of Johnny','FontSize',12);
axis([0 256 0 2500]);
set(gca,'XTick',[0:16:256]);
set(gca,'YTick',[0:250:2500]);
print -deps MJohnnyEQHistCrop.eps;
print -dpdf MJohnnyEQHist.pdf;


%
% HistEQ.m
%
%   Perform histogram equalization on a gray scale byte-per-pixel image.
%
%   Usage:
%   >> K = HistEQ(x);
%
%   Input parameters:
%     x          double array, holds the original image
%
%   Output parameters:
%     K          double array, holds the equalized image
%
% 4/3/03 jph
% 2/3/15: improvements for faster run time. jph
%

function [K] = HistEQ(x)

%
% Compute the histogram of the input image
%
InputHist = sum(hist(x,0:255)');

%
% Compute the normalized histogram of the input image
%
[Nrows,Ncols] = size(x);
Npix = Nrows * Ncols;
One_on_Npix = 1 / Npix;
p_I = InputHist * One_on_Npix;

%
% Compute the CDF (distribution)
%
P_I = zeros(1,256);
P_I(1) = p_I(1);
for i=2:256
  P_I(i) = P_I(i-1) + p_I(i);
```

```
end

%
% Compute the intermediate image J.  It is the CDF of x.
%
J = zeros(Nrows,Ncols);
for row=1:Nrows
  for col=1:Ncols
    J(row,col) = P_I(x(row,col)+1);
  end
end

%
% The output image K is obtained by applying a
%     full-scale contrast stretch to J.
%
K = stretch(J);
```