Appendix A: MATLAB Functions

Matlab	Description	function	Usage Location
ddencmp()	Returns the Default threshold values for denoising and compression	[THR,SORH,KEEPAPP]= **ddencmp(**IN1,'wv',X**)**	Wavelet toolbox
double()	Convert data to double precision	B = **double(**A**)**	Image processing toolbox
dwt2()	Performs single level discrete 2-D wavelet transform	[cA,cH,cV,cD] = **dwt2(**X,'*wname*'**)**	Wavelet toolbox
eig	**()** Returns matrix eigenvalues and eigen vectors	lambda = **eig(**A**)**	Symbolic Math toolbox
idwt2()	Performs single level inverse discrete 2-D wavelet transform	X =**idwt2(**cA,cH,cV,cD,'*wname*'**)**	Wavelet toolbox
imnoise()	Adds noise to an image	J = **imnoise(**I,*type***)**	Image processing toolbox
imread()	Read image from graphics files	[A] = **imread(**filename**)**	Image processing toolbox
imshow()	Displays an image	**imshow(**A**)**	Image processing toolbox
imwrite()	Writes image to graphics file	**imwrite(**A,filename**)**	Image processing toolbox

medfilt2()	Performs two - dimensional median filtering	B= m**edfilt2(**A,[m n]**)**	Image processing toolbox
uint8()	Converts data to unsigned 8-	B = **uint8(**A**)**	Image processing
wavedec2()	Performs multilevel 2-D wavelet decomposition	[C,S] = **wavedec2(**X,N,'*wname*'**)**	Wavelet toolbox
wdcbm2()	Returns 2-D threshold value based on SureShrink	[THR,NKEEP] = **wdcbm2(**C,S,ALPHA**)**	Wavelet toolbox
wdencmp() Wavelet toolbox	Performs De-noising or compression using wavelets	[XD] = **wdencmp(**'gbl',X,'*wname*',N,THR,SOR H,KEEPAPP**)** [XC] = **wdencmp(**'lvd',C,L,'*wname*',N,THR,SO RH**)**	Wavelet toolbox
wthresh()	Performs hard or soft thresholding	Y = **wthresh(**X,SORH,T**)**	Wavelet toolbox
wthrmngr()	De-noising using level dependent thresholds	THR = WTHRMNGR('dw2ddenoLVL','penalhi',C,S,ALFA) ALFA must be such that 2.5 < ALFA < 10**)**	Wavelet toolbox

Appendix A: Latest Versions MATLAB

Latest versions of MATLAB are listed below.

MATLAB 8.5	R2015a	33	1.7.0_60	2015	March 5, 2015
MATLAB 8.6	R2015b	34	1.7.0_60		September 3, 2015
MATLAB 9.0	R2016a	35	1.7.0_60	2016	March 3, 2016

```
% Call the Sobel () function with the image path as the argument.
Sobel ('Darkly.jpg ');
% Sobel ('water2.jpg ');
```

```
% We can first read in the source image using the imread ()
% function . Since we are only operating on grayscale images ,
% we will take the gray transformation using rgb2gray ().
% Finally , we want to define our Sobel matrices , and use
% the built in convolution function conv () to convolve the
% source image with the Sobel kernels . For the sake of code
% reuse with multiple images , we will incorporate these
% operations into a function :
%
function Sobel (im)

% Read in the image and convert to gray
orig      = imread (im);
grayscale = rgb2gray ( orig );

% Display the original and gray image
figure (1);
imshow ( grayscale );
figure (2);
imshow ( orig );

% Define the Sobel kernels
k_v = [-1 0 1; -2 0 2; -1 0 1];
k_h = [1 2 1; 0 0 0; -1 -2 -1];

% Convolve the gray image with Sobel kernels , store result in
M1 = conv2 ( double ( grayscale ), double (k_v ));
M2 = conv2 ( double ( grayscale ), double (k_h ));

% Display the horizontal edges and vertical edges separately
figure (3);
imshow ( abs (M1), []) ;
figure (4);
imshow ( abs (M2), []) ;

% Display the normalized vertical and horizontal edges
figure (5);
imshow (( M1 .^2+ M2 .^2) .^0.5 , []);
end
```

sobel2.m and sobel.m MATLAB Convolution Denoised Routine [36]

```
N = 512;  k = 9;
pupil = make_pupil(N,2*k);    % construct the pupil function
rmax = N/(4*k);
w = zeros(N,N);               % perfect wavefront
z = psf(pupil,w);             % generate the spread function
ncen = 1+N/2;
zp = z(ncen:ncen+30,ncen);    % take the central portion
xp = (0:30)/(2*k);            % generate x-coordinates
% calculate exact function
xe = linspace(0,30/(2*k),500);
ze = somb(2*xe);
ze = ze.*ze;
% plot the results
plot(xe,ze,xp,zp,'o');
xlabel('radius/\epsilon_o');
ylabel('irradiance');
% generate and save the images.
% Use logrithmic transformation for output image
imshow(log_image(z,5));
imwrite(255*log_image(z,5),gray(255),'airy.bmp');
imwrite(z,'airy.pgm','pgm','Encoding','ASCII');
imwrite(255*pupil,gray(255),'pupil.bmp');
```

MATLAB routine airy.m to calculate
Airy Point Spread Function (PSF)
[46]

```
clear;
clc;
clear all;
close all;

display('select the image');
display('           1:lena.png');
display('           2:barbara.png');
display('           3:peppers256.png');
display('           4:cameraman.jpg');
display('           ');

ss1=input('enter your choice:   ');
switch ss1
    case 1
       f=imread('lena.png');
    case 2
        f=imread('barbara.png');
    case 3
        f=imread('peppers256.png');
    case 4
        f=imread('cameraman.jpg');
end

subplot(2,2,1), imshow(f);title('original image');
imwrite(f,'x1.jpg','jpeg')

display('enter the type of noise:');
display('    1    for salt & pepper');
display('    2    for gaussian');
display('    3    for poisson');
display('    4    for speckle');

ud=input('enter the value:');

switch ud

    case 1
        display('enter the % of noise(Ex:0.2)');
        ud1=input('pls enter:     ');
        g=imnoise(f,'salt & pepper',ud1);

    case 2
    display('enter the noise varience:  ');
    va=input('enter between 0.01 to 0.09:   ');
    g=imnoise(f,'gaussian',0,va);

    case 3
        g=imnoise(f,'poisson');

    case 4
    display('enter the varience of noise(Ex:0.02)');
    ud1=input('pls enter:     ');
    g=imnoise(f,'speckle',ud1);
end

subplot(2,2,2),imshow(g);title('noisy image');
imwrite(g,'x2.jpg','jpeg');

x=g;
% Use wdencmp for image de-noising.
% find default values (see ddencmp).
[thr,sorh,keepapp] = ddencmp('den','wv',x);

display('');
display('select wavelet');
display('enter 1 for  haar    wavelet');
display('enter 2 for  db4     wavelet');
display('enter 3 for  sym4    wavelet');
display('enter 4 for  bior6.8 wavelet');
display('press any key to quit');
display('');

ww=input('enter your choice:    ');
switch ww
    case 1
        wv='haar';
    case 2
```

MATLAB Denoised
denoise.m Program [9]

```
        wv='db4' ;
    case 3
        wv='sym4';
    case 4
       wv='bior6.8';
    otherwise
        quit;
end
display('');
display('enter 1 for soft thresholding');
display('enter 2 for hard thresholding');
display('enter 3 for bayes soft thresholding');
sorh=input('sorh:   ');

display('enter the level of decomposition');
level=input(' enter 1 or 2 :    ');

switch sorh
    case 1
        sorh='s';
        xd = wdencmp('gbl',x,wv,level,thr,sorh,keepapp);
    case 2
        sorh='h';
        xd = wdencmp('gbl',x,wv,level,thr,sorh,keepapp);
    case 3
        %%%%%%%%%%%%%%%%%%%%%
       clear all;

%Denoising using Bayes soft thresholding

%Note: Figure window 1 displays the original image, fig 2 the noisy img
%fig 3 denoised img by bayes soft thresholding

%Reading the image
pic=f;

%Define the Noise Variance and adding Gaussian noise
%While using 'imnoise' the pixel values(0 to 255) are converted to double
%So variance also has to be suitably converted
%So variance also has to be suitably converted
sig=15;
V=(sig/256)^2;
npic=g;

%Define the type of wavelet(filterbank) used and the number of scales
filtertype=wv;
levels=level;

%Doing the wavelet decomposition
[C,S]=wavedec2(npic,levels,filtertype);

st=(S(1,1)^2)+1;
bayesC=[C(1:st-1),zeros(1,length(st:1:length(C)))];
var=length(C)-S(size(S,1)-1,1)^2+1;

%Calculating sigmahat
sigmahat=median(abs(C(var:length(C))))/0.6745;

for jj=2:size(S,1)-1
    %for the H detail coefficients
    coefh=C(st:st+S(jj,1)^2-1);
    thr=bayes(coefh,sigmahat);
    bayesC(st:st+S(jj,1)^2-1)=sthresh(coefh,thr);
    st=st+S(jj,1)^2;

    % for the V detail coefficients
    coefv=C(st:st+S(jj,1)^2-1);
    thr=bayes(coefv,sigmahat);
    bayesC(st:st+S(jj,1)^2-1)=sthresh(coefv,thr);
    st=st+S(jj,1)^2;

    %for Diag detail coefficients
    coefd=C(st:st+S(jj,1)^2-1);
    thr=bayes(coefd,sigmahat);
    bayesC(st:st+S(jj,1)^2-1)=sthresh(coefd,thr);
    st=st+S(jj,1)^2;
end


%Reconstructing the image from the Bayes-thresholded wavelet coefficients
bayespic=waverec2(bayesC,S,filtertype);
xd=bayespic;
%Displaying the Bayes-denoised image

end

% de-noise image using global thresholding option.
[c,s]=wavefast(g,level,wv);
subplot(2,2,3),wave2gray(c,s,8);title('decomposed structure');

subplot(2,2,4),xd=uint8(xd);
imshow(xd);title('denoised image');

imwrite(xd,'x3.jpg','jpeg');

ff=im2double(f);xdd=im2double(xd);orig=im2double(f);gg=im2double(g);
display('     ');
display('reference: MSE  ratio');
display('reference: MAE  ratio');
display('reference: PSNR ratio');
display('     ');
mse=compare11(ff,xdd)
mae=maer(orig,xdd)
snr=upsnr(xdd,gg)
```

MATLAB Denoised
denoise.m Program [9]

Medical Glossary

CAT or CT SCAN	-Computed Tomography.
CT Head wo contrast Standard	-Standard MRI is the standard in medical diagnostic standard testing. There is no radiation involved and the test results are comprehensive.
CT Head wo contrast Scout	-Scout film is a preliminary film taken of a body region. Scout is taken before a definitive imaging study- e.g., a scout film of the chest before a CT. "Scouts" serve to establish a baseline and may be used before performing angiography, CT, or MRI.
MRI Brain Ax T1 SE	-T1 weighted MR image of the brain shows cortex, white and grey matter, third and lateral ventricles, putamen, frontal sinus and superior sagittal sinus.
MRA Carotid 3 – Plane Fiesta	-Plane: 3-Plane : Pulse Sequence: Fiesta.
MRA Carotid Left 2D TOF Rotate	-2D TOF imaging in 2D acquisition, time-of-flight imaging uses a set of fine slices that are stacked up to, Reconstruct a pseudo-volume. The advantage of fine slices are better sensitivity to slow flows.
T1	-Weighted MR image of the brain shows eyeballs with optic nerve, medulla, vermis, and temporal lobes with hippocampal regions.
T2	- Weighted MR image of the head shows maxillary sinus, nasal septum, clivus, inner ear, medulla, and cerebellum.
T1/w	-Scans with Contrast.
T1/wo	-Scans without Contrast.

T2/FLAIR	-Images show the total amount of scar from MS from its onset. The pictures show both old and new inflammation.
XR Chest AP Portable	-X-ray Chest Anteroposterior Position.

Appendix C: Programs in C Language

```
/* An Image Processing Algoritthm */
#define MAX
#include "fftlib.h"

int
main ()
{
    FILE *ptr_file;
    ptr_file=fopen("output.txt","w");
        int n=0;
        int nr, nc, again, radius;
         int filter;
         int order;
        IMAGE im1, p;
        COMPLEX_IMAGE fft1, fft2;
        float x, xi, H, K, t;
        float H1, H2 ,H3 , H4, sigma;
        int offset;
        int i,j,k, hc, vc, width, dist;

        printf ("Filter = 0  Median         Filter\n");
        printf ("Filter = 1  Low  Pass      Filter\n");
        printf ("Filter = 2  High Pass      Filter\n");
        printf ("Filter = 3  Wiener         Filter\n");
        printf ("Filter = 4  Butterworth  Lo Filter\n");
        printf ("Filter = 5  Butterworth  Hi Filter\n");
        printf ("Filter = 6  Exponential  Lo Filter\n");
        printf ("Filter = 7  Gaussian      Filter\n");
        printf ("Filter = 8  Adaptive Wiener Filter\n");

      scanf ("%d", &filter);


      if (filter ==0)
           goto MEDIAN;
      if (filter ==1)
      goto LowHigh;
      if (filter ==2)
```

Appendix C: Programs in C Language

```
goto LowHigh;
if (filter ==3)
goto Wiener;
if (filter ==4)
goto ButterLO;
if (filter ==5)
goto ButterHI;
if (filter ==6)
      goto ExponLOW;
      if (filter ==7)
      goto Gauss;
       if (filter ==8)
      goto Adaptive;
```

LowHigh:

```
      im1 = Input_PBM ("lena_noise.pgm");

      nr = im1->info->nr; nc = im1->info->nc;


      printf ("Read Radius R\n");

    scanf ("%d", &radius);

      normalize_set();
      image_fftoc (im1, &fft1);
      normalize_clear();

/* Clear pixels in the specified radius of the center */
      hc = nc/2; vc = nr/2;
      for (i=0; i<nr; i++)
        for (j=0; j<nc; j++)
        {
          dist = (float)sqrt((double)((i-vc)*(i-vc) + (j-hc)*(j-hc)));

          if ( (dist > radius) && (filter==1) )
          {
                fft1[i][j]    = 0.0;
                fft1[i][j+nc] = 0.0;
          }
```

```
        if ( (dist <= radius) && (filter==2) )
        {
                fft1[i][j]    = 0.0;
                fft1[i][j+nc] = 0.0;
        }
       }

     image_fftinv (fft1, &fft2);
     realtoint (fft2, 0);
     for (i=0; i<nr; i++)
      for (j=0; j<nc; j++)
         im1->data[i][j] = (int)fft2[i][j];
     Output_PBM (im1, "lowhigh.pgm");
   goto ENDEND
```

Wiener:

```
     im1 = Input_PBM ("lena_noise.pgm");
     p   = Input_PBM ("PSF.pgm");

     nr  = im1->info->nr; nc = im1->info->nc;

     image_fftoc (im1, &fft1);
     normalize_set ();
     image_fftoc (p,   &fft2);
     normalize_clear ();

     printf ("Enter K: ");
     scanf ("%f", &K);

     for (i=0; i<nr; i++)
      for (j=0; j<nc; j++)
      {
        x = fft2[i][j];  xi = fft2[i][j+nc];
        H = pix_cnorm (x, xi);
        if (H == 0.0)
        {
          fft1[i][j] = fft1[i][j+n] = 0.0;
```

```
        continue;
      }
      t = H/(H+K);
      cdiv (t, 0.0, &x, &xi);
      cprod (fft1[i][j], fft1[i][j+nc], &x, &xi);
      fft1[i][j] = x; fft1[i][j+nc] = xi;
    }

  freecomplex (fft2); fft2 = (COMPLEX_IMAGE)0;

  image_fftinvoc (fft1, &fft2);
  freecomplex (fft1); fft1 = (COMPLEX_IMAGE)0;

  filt_toint (fft2, p, 0);
  Output_PBM (p,   "wiener.pgm");

  printf ("End of Wiener\n");
   goto ENDEND;
```

ButterLO:

```
  im1 = Input_PBM ("lena_noise.pgm");
  p   = Input_PBM ("PSF.pgm");

  nr = im1->info->nr; nc = im1->info->nc;

  printf ("Read CutOff Radius R\n");
  scanf ("%d", &radius);

  printf ("Read Order of Butter\n");
  scanf ("%d", &order);

  normalize_set();
  image_fftoc (im1, &fft1);
  normalize_clear();

/* Clear pixels in the specified radius of the center */
  hc = nc/2; vc = nr/2;
  for (i=0; i<nr; i++)
    for (j=0; j<nc; j++)
    {
```

```
        dist = (float)sqrt((double)((i-vc)*(i-vc) + (j-hc)*(j-hc)));

       H=1.0/(1.0+0.414*((dist/radius)^(2*order)));

              fft1[i][j]    = fft1[i][j]*H;
              fft1[i][j+nc] = fft1[i][j+nc]*H;
      }

    image_fftinv (fft1, &fft2);
    realtoint (fft2, 0);
    for (i=0; i<nr; i++)
      for (j=0; j<nc; j++)
         im1->data[i][j] = (int)fft2[i][j];
    Output_PBM (im1, "butterlow.pgm");
    goto ENDEND;
```

ButterHI:

```
    im1 = Input_PBM ("lena_noise.pgm");
    p   = Input_PBM ("PSF.pgm");

    nr = im1->info->nr; nc = im1->info->nc;

    printf ("Read CutOff Radius R\n");
    scanf ("%d", &radius);

    printf ("Read Order of Butter\n");
    scanf ("%d", &order);

    normalize_set();
    image_fftoc (im1, &fft1);
    normalize_clear();

/* Clear pixels in the specified radius of the center */
    hc = nc/2; vc = nr/2;
    for (i=0; i<nr; i++)
      for (j=0; j<nc; j++)
      {
        dist = (float)sqrt((double)((i-vc)*(i-vc) + (j-hc)*(j-hc)));

        if (dist == 0) {
              H=1.00;
        }
```

```
        else {
           H=1.0/(1.0+0.414*((radius/dist)^(2*order)));
        }

                fft1[i][j]    = fft1[i][j]*H;
                fft1[i][j+nc] = fft1[i][j+nc]*H;
          }

        image_fftinv (fft1, &fft2);
        realtoint (fft2, 0);
        for (i=0; i<nr; i++)
          for (j=0; j<nc; j++)
             im1->data[i][j] = (int)fft2[i][j];
        Output_PBM (im1, "butterhigh.pgm");
      goto ENDEND;
```

ExponLOW:

```
        im1 = Input_PBM ("lena_noise.pgm");

        nr = im1->info->nr; nc = im1->info->nc;

        printf ("Read CutOff Radius R\n");
        scanf ("%d", &radius);

        printf ("Read Order of Filter\n");
        scanf ("%d", &order);

        printf ("Read OFFSET   Filter\n");
        scanf ("%d", &offset);

        normalize_set();
        image_fftoc (im1, &fft1);
        normalize_clear();

/* Clear pixels in the specified radius of the center */
        hc = nc/2; vc = nr/2;
        for (i=0; i<nr; i++)
          for (j=0; j<nc; j++)
          {
            dist = (float)sqrt((double)((i-vc)*(i-vc) + (j-hc)*(j-hc)));
```

```
            if (dist <= 0) {
                  H=1.00;
            }
          else {
            H=exp(-(((dist - offset)/radius)^order));
          }

                fft1[i][j]    = fft1[i][j]*H;
                fft1[i][j+nc] = fft1[i][j+nc]*H;
             }

          image_fftinv (fft1, &fft2);
          realtoint (fft2, 0);
          for (i=0; i<nr; i++)
            for (j=0; j<nc; j++)
               im1->data[i][j] = (int)fft2[i][j];
          Output_PBM (im1, "exponentiallow.pgm");
          goto ENDEND;
```

Gauss:

```
          im1 = Input_PBM ("lena_noise.pgm");

          nr = im1->info->nr; nc = im1->info->nc;

          printf ("Read Radius\n");
        scanf ("%d", &radius);

          normalize_set();
          image_fftoc (im1, &fft1);
          normalize_clear();

/* Clear pixels in the specified radius of the center */
          hc = nc/2; vc = nr/2;
          for (i=0; i<nr; i++)
            for (j=0; j<nc; j++)
            {

    dist = (float)((double)((i-vc)*(i-vc) + (j-hc)*(j-hc)));

             H =exp(-dist / 2*radius*radius);
             H1=exp(1.0);
```

```
            H4=exp(dist / 2*radius*radius);
            H2=2.0*3.14159*sigma*sigma;
            H3=H1/H2;
                    fft1[i][j]    = fft1[i][j]*H;
                    fft1[i][j+nc] = fft1[i][j+nc]*H;
            fprintf(ptr_file,"%12.4e\n",H);
           }

         image_fftinv (fft1, &fft2);
         realtoint (fft2, 0);
         for (i=0; i<nr; i++)
           for (j=0; j<nc; j++)
              im1->data[i][j] = (int)fft2[i][j];
         Output_PBM (im1, "gauss.pgm");
         Goto ENDEND;

ENDEND:
         printf ("End of Program\n");
```

```
/* End of C Language Program*/
```