Apr 26, 97 12:48 **binomialFilter.m** Page 1/1

```
% KERNEL = binomialFilter(size)
%
% Returns a vector of binomial coefficients of order (size−1) .

% Eero Simoncelli, 2/97.

function [kernel] = binomialFilter(sz)

if (sz < 2)
  error('size argument must be larger than 1');
end

kernel = [0.5 0.5]';

for n=1:sz−2
  kernel = conv([0.5 0.5]', kernel);
end
```

Aug 06, 03 18:04 **blurDn.m** Page 1/1

```
% RES = blurDn(IM, LEVELS, FILT)
%
% Blur and downsample an image.  The blurring is done with filter
% kernel specified by FILT (default = 'binom5'), which can be a string
% (to be passed to namedFilter), a vector (applied separably as a 1D
% convolution kernel in X and Y), or a matrix (applied as a 2D
% convolution kernel).  The downsampling is always by 2 in each
% direction.
%
% The procedure is applied recursively LEVELS times (default=1).

% Eero Simoncelli, 3/97.

function res = blurDn(im, nlevs, filt)

%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
%% OPTIONAL ARGS:

if (exist('nlevs') ~= 1)
  nlevs = 1;
end

if (exist('filt') ~= 1)
  filt = 'binom5';
end

%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

if isstr(filt)
  filt = namedFilter(filt);
end

filt = filt/sum(filt(:));

if nlevs > 1
  im = blurDn(im,nlevs−1,filt);
end

if (nlevs >= 1)
  if (any(size(im)==1))
    if (~any(size(filt)==1))
      error('Cant  apply 2D filter to 1D signal');
    end
    if (size(im,2)==1)
      filt = filt(:);
    else
      filt = filt(:)';
    end
    res = corrDn(im,filt,'reflect1',(size(im)~=1)+1);
  elseif (any(size(filt)==1))
    filt = filt(:);
    res = corrDn(im,filt,'reflect1',[2 1]);
    res = corrDn(res,filt','reflect1',[1 2]);
  else
    res = corrDn(im,filt,'reflect1',[2 2]);
  end
else
  res = im;
end
```

```
% [PYR, INDICES] = buildGpyr(IM, HEIGHT, FILT, EDGES)
%
% Construct a Gaussian pyramid on matrix IM.
%
% HEIGHT (optional) specifies the number of pyramid levels to build. Default
% is 1+maxPyrHt(size(IM),size(FILT)).
% You can also specify 'auto' to use this value.
%
% FILT (optional) can be a string naming a standard filter (see
% namedFilter), or a vector which will be used for (separable)
% convolution.  Default = 'binom5'.  EDGES specifies edge-handling, and
% defaults to 'reflect1' (see corrDn).
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.

% Eero Simoncelli, 6/96.

function [pyr,pind] = buildGpyr(im, ht, filt, edges)

if (nargin < 1)
  error('First argument (IM) is required');
end

im_sz = size(im);

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('filt') ~= 1)
  filt = 'binom5';
end

if isstr(filt)
  filt = namedFilter(filt);
end

if ( (size(filt,1) > 1) & (size(filt,2) > 1) )
  error('FILT should be a 1D filter (i.e., a vector)');
else
  filt = filt(:);
end

max_ht = 1 + maxPyrHt(im_sz, size(filt,1));
if ( (exist('ht') ~= 1) | (ht == 'auto') )
  ht = max_ht;
else
  if (ht > max_ht)
    error(sprintf('Cannot build pyramid higher than %d levels.',max_ht));
  end
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end

%------------------------------------------------------------

if (ht <= 1)

  pyr = im(:);
  pind = im_sz;

else

  if (im_sz(2) == 1)
    lo2 = corrDn(im, filt, edges, [2 1], [1 1]);
  elseif (im_sz(1) == 1)
    lo2 = corrDn(im, filt', edges, [1 2], [1 1]);
  else
    lo = corrDn(im, filt', edges, [1 2], [1 1]);
    lo2 = corrDn(lo, filt, edges, [2 1], [1 1]);
  end

  [npyr,nind] = buildGpyr(lo2, ht-1, filt, edges);

  pyr = [im(:); npyr];
  pind = [im_sz; nind];

end
```



```
% [PYR, INDICES] = buildLpyr(IM, HEIGHT, FILT1, FILT2, EDGES)
%
% Construct a Laplacian pyramid on matrix (or vector) IM.
%
% HEIGHT (optional) specifies the number of pyramid levels to build. Default
% is 1+maxPyrHt(size(IM),size(FILT)).  You can also specify 'auto' to
% use this value.
%
% FILT1 (optional) can be a string naming a standard filter (see
% namedFilter), or a vector which will be used for (separable)
% convolution.  Default = 'binom5'.  FILT2 specifies the "expansion"
% filter (default = filt1).  EDGES specifies edge-handling, and
% defaults to 'reflect1' (see corrDn).
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.

% Eero Simoncelli, 6/96.

function [pyr,pind] = buildLpyr(im, ht, filt1, filt2, edges)

if (nargin < 1)
  error('First argument (IM) is required');
end

im_sz = size(im);

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('filt1') ~= 1)
  filt1 = 'binom5';
end

if isstr(filt1)
  filt1 = namedFilter(filt1);
end

if ( (size(filt1,1) > 1) & (size(filt1,2) > 1) )
  error('FILT1 should be a 1D filter (i.e., a vector)');
else
  filt1 = filt1(:);
end

if (exist('filt2') ~= 1)
  filt2 = filt1;
end

if isstr(filt2)
  filt2 = namedFilter(filt2);
end

if ( (size(filt2,1) > 1) & (size(filt2,2) > 1) )
  error('FILT2 should be a 1D filter (i.e., a vector)');
else
  filt2 = filt2(:);
end

max_ht = 1 + maxPyrHt(im_sz, max(size(filt1,1), size(filt2,1)));
if ( (exist('ht') ~= 1) | (ht == 'auto') )
  ht = max_ht;
else
  if (ht > max_ht)
    error(sprintf('Cannot build pyramid higher than %d levels.',max_ht));
  end
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end

%------------------------------------------------------------

if (ht <= 1)

  pyr = im(:);
  pind = im_sz;

else

  if (im_sz(2) == 1)
    lo2 = corrDn(im, filt1, edges, [2 1], [1 1]);
  elseif (im_sz(1) == 1)
    lo2 = corrDn(im, filt1', edges, [1 2], [1 1]);
  else
    lo = corrDn(im, filt1', edges, [1 2], [1 1]);
    int_sz = size(lo);
    lo2 = corrDn(lo, filt1, edges, [2 1], [1 1]);
  end

  [npyr,nind] = buildLpyr(lo2, ht-1, filt1, filt2, edges);

  if (im_sz(1) == 1)
    hi2 = upConv(lo2, filt2', edges, [1 2], [1 1], im_sz);
  elseif (im_sz(2) == 1)
    hi2 = upConv(lo2, filt2, edges, [2 1], [1 1], im_sz);
  else
    hi = upConv(lo2, filt2, edges, [2 1], [1 1], int_sz);
    hi2 = upConv(hi, filt2', edges, [1 2], [1 1], im_sz);
  end

  hi2 = im - hi2;

  pyr = [hi2(:); npyr];
  pind = [im_sz; nind];

end
```

Aug 28, 02 21:57 **buildSFpyrLevs.m** Page 1/1

```
% [PYR, INDICES] = buildSFpyrLevs(LODFT, LOGRAD, XRCOS, YRCOS, ANGLE, HEIGHT, NB
ANDS)
%
% Recursive function for constructing levels of a steerable pyramid.  This
% is called by buildSFpyr, and is not usually called directly.

% Eero Simoncelli, 5/97.

function [pyr,pind] = buildSFpyrLevs(lodft,log_rad,Xrcos,Yrcos,angle,ht,nbands);

if (ht <= 0)

  lo0 = ifft2(ifftshift(lodft));
  pyr = real(lo0(:));
  pind = size(lo0);

else

  bands = zeros(prod(size(lodft)), nbands);
  bind = zeros(nbands,2);

%  log_rad = log_rad + 1;
  Xrcos = Xrcos - log2(2);  % shift origin of lut by 1 octave.

  lutsize = 1024;
  Xcosn = pi*[-(2*lutsize+1):(lutsize+1)]/lutsize;  % [-2*pi:pi]
  order = nbands-1;
  %% divide by sqrt(sum_(n=0)^(N-1)  cos(pi*n/N)^(2(N-1)) )
  %% Thanks to Patrick Teo for writing this out :)
  const = (2^(2*order))*(factorial(order)^2)/(nbands*factorial(2*order));
  Ycosn = sqrt(const) * (cos(Xcosn)).^order;
  himask = pointOp(log_rad, Yrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);

  for b = 1:nbands
    anglemask = pointOp(angle, Ycosn, Xcosn(1)+pi*(b-1)/nbands, Xcosn(2)-Xcosn(1
));
    banddft = ((-sqrt(-1))^(nbands-1)) .* lodft .* anglemask .* himask;
    band = ifft2(ifftshift(banddft));

    bands(:,b) = real(band(:));
    bind(b,:)  = size(band);
  end

  dims = size(lodft);
  ctr = ceil((dims+0.5)/2);
  lodims = ceil((dims-0.5)/2);
  loctr = ceil((lodims+0.5)/2);
  lostart = ctr-loctr+1;
  loend = lostart+lodims-1;

  log_rad = log_rad(lostart(1):loend(1),lostart(2):loend(2));
  angle = angle(lostart(1):loend(1),lostart(2):loend(2));
  lodft = lodft(lostart(1):loend(1),lostart(2):loend(2));
  YIrcos = abs(sqrt(1.0 - Yrcos.^2));
  lomask = pointOp(log_rad, YIrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);

  lodft = lomask .* lodft;

  [npyr,nind] = buildSFpyrLevs(lodft, log_rad, Xrcos, Yrcos, angle, ht-1, nbands
);

  pyr = [bands(:); npyr];
  pind = [bind; nind];

end
```

Aug 14, 03 15:49 **buildSFpyr.m** Page 1/1

```
% [PYR, INDICES, STEERMTX, HARMONICS] = buildSFpyr(IM, HEIGHT, ORDER, TWIDTH)
%
% Construct a steerable pyramid on matrix IM, in the Fourier domain.
% This is similar to buildSpyr, except that:
%
%    + Reconstruction is exact (within floating point errors)
%    + It can produce any number of orientation bands.
%    - Typically slower, especially for non-power-of-two sizes.
%    - Boundary-handling is circular.
%
% HEIGHT (optional) specifies the number of pyramid levels to build. Default
% is maxPyrHt(size(IM),size(FILT));
%
% The squared radial functions tile the Fourier plane, with a raised-cosine
% falloff.  Angular functions are cos(theta-k\pi/(K+1))^K, where K is
% the ORDER (one less than the number of orientation bands, default= 3).
%
% TWIDTH is the width of the transition region of the radial lowpass
% function, in octaves (default = 1, which gives a raised cosine for
% the bandpass filters).
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.
% See the function STEER for a description of STEERMTX and HARMONICS.

% Eero Simoncelli, 5/97.
% See http://www.cis.upenn.edu/~eero/steerpyr.html for more
% information about the Steerable Pyramid image decomposition.

function [pyr,pind,steermtx,harmonics] = buildSFpyr(im, ht, order, twidth)

%-----------------------------------------------------------------
%% DEFAULTS:

max_ht = floor(log2(min(size(im)))) - 2;

if (exist('ht') ~= 1)
  ht = max_ht;
else
  if (ht > max_ht)
    error(sprintf('Cannot build pyramid higher than %d levels.',max_ht));
  end
end

if (exist('order') ~= 1)
  order = 3;
elseif ((order > 15)  | (order < 0))
  fprintf(1,'Warning: ORDER must be an integer in the range [0,15]. Truncating.\
n');
  order = min(max(order,0),15);
else
  order = round(order);
end
nbands = order+1;

if (exist('twidth') ~= 1)
  twidth = 1;
elseif (twidth <= 0)
  fprintf(1,'Warning: TWIDTH must be positive.  Setting to 1.\n');
  twidth = 1;
end

%-----------------------------------------------------------------
%% Steering stuff:

if (mod((nbands),2) == 0)
  harmonics = [0:(nbands/2)-1]'*2 + 1;
else
  harmonics = [0:(nbands-1)/2]'*2;
end

steermtx = steer2HarmMtx(harmonics, pi*[0:nbands-1]/nbands, 'even');

%-----------------------------------------------------------------

dims = size(im);
ctr = ceil((dims+0.5)/2);

[xramp,yramp] = meshgrid( ([1:dims(2)]-ctr(2))./(dims(2)/2), ...
    ([1:dims(1)]-ctr(1))./(dims(1)/2) );
angle = atan2(yramp,xramp);
log_rad = sqrt(xramp.^2 + yramp.^2);
log_rad(ctr(1),ctr(2)) =  log_rad(ctr(1),ctr(2)-1);
log_rad  = log2(log_rad);

%% Radial transition function (a raised cosine in log-frequency):
[Xrcos,Yrcos] = rcosFn(twidth,(-twidth/2),[0 1]);
Yrcos = sqrt(Yrcos);

YIrcos = sqrt(1.0 - Yrcos.^2);
lo0mask = pointOp(log_rad, YIrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);
imdft = fftshift(fft2(im));
lo0dft =  imdft .* lo0mask;

[pyr,pind] = buildSFpyrLevs(lo0dft, log_rad, Xrcos, Yrcos, angle, ht, nbands);

hi0mask = pointOp(log_rad, Yrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);
hi0dft =  imdft .* hi0mask;
hi0 = ifft2(ifftshift(hi0dft));

pyr = [real(hi0(:)) ; pyr];
pind = [size(hi0); pind];
```

```
% [PYR, INDICES] = buildSpyrLevs(LOIM, HEIGHT, LOFILT, BFILTS, EDGES)
%
% Recursive function for constructing levels of a steerable pyramid.  This
% is called by buildSpyr, and is not usually called directly.

% Eero Simoncelli, 6/96.

function [pyr,pind] = buildSpyrLevs(lo0,ht,lofilt,bfilts,edges);

if (ht <= 0)

  pyr = lo0(:);
  pind = size(lo0);

else

  % Assume square filters:
  bfiltsz =  round(sqrt(size(bfilts,1)));

  bands = zeros(prod(size(lo0)),size(bfilts,2));
  bind = zeros(size(bfilts,2),2);

  for b = 1:size(bfilts,2)
    filt = reshape(bfilts(:,b),bfiltsz,bfiltsz);
    band = corrDn(lo0, filt, edges);
    bands(:,b) = band(:);
    bind(b,:)  = size(band);
  end

  lo = corrDn(lo0, lofilt, edges, [2 2], [1 1]);

  [npyr,nind] = buildSpyrLevs(lo, ht-1, lofilt, bfilts, edges);

  pyr = [bands(:); npyr];
  pind = [bind; nind];

end
```



```
% [PYR, INDICES, STEERMTX, HARMONICS] = buildSpyr(IM, HEIGHT, FILTFILE, EDGES)
%
% Construct a steerable pyramid on matrix IM.
%
% HEIGHT (optional) specifies the number of pyramid levels to build. Default
% is maxPyrHt(size(IM),size(FILT)).
% You can also specify 'auto' to use this value.
%
% FILTFILE (optional) should be a string referring to an m-file that
% returns the rfilters.  (examples: 'sp0Filters', 'sp1Filters',
% 'sp3Filters','sp5Filters'.  default = 'sp1Filters'). EDGES specifies
% edge-handling, and defaults to 'reflect1' (see corrDn).
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.
% See the function STEER for a description of STEERMTX and HARMONICS.

% Eero Simoncelli, 6/96.
% See http://www.cis.upenn.edu/~eero/steerpyr.html for more
% information about the Steerable Pyramid image decomposition.

function [pyr,pind,steermtx,harmonics] = buildSpyr(im, ht, filtfile, edges)

%-----------------------------------------------------------------
%% DEFAULTS:

if (exist('filtfile') ~= 1)
  filtfile = 'sp1Filters';
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end

if (isstr(filtfile) & (exist(filtfile) == 2))
   [lo0filt,hi0filt,lofilt,bfilts,steermtx,harmonics] = eval(filtfile);
else
  fprintf(1,'\nUse buildSFpyr for pyramids with arbitrary numbers of orientation
 bands.\n');
  error('FILTFILE argument must be the name of an M-file containing SPYR filters
.');
end

max_ht = maxPyrHt(size(im), size(lofilt,1));
if ( (exist('ht') ~= 1) | (ht == 'auto') )
  ht = max_ht;
else
  if (ht > max_ht)
    error(sprintf('Cannot build pyramid higher than %d levels.',max_ht));
  end
end

%-----------------------------------------------------------------

hi0 = corrDn(im, hi0filt, edges);
lo0 = corrDn(im, lo0filt, edges);

[pyr,pind] = buildSpyrLevs(lo0, ht, lofilt, bfilts, edges);

pyr = [hi0(:) ; pyr];
pind = [size(hi0); pind];
```

Apr 16, 98 17:43 **buildWpyr.m** Page 1/1

```
% [PYR, INDICES] = buildWpyr(IM, HEIGHT, FILT, EDGES)
%
% Construct a separable orthonormal QMF/wavelet pyramid on matrix (or vector) IM
.
%
% HEIGHT (optional) specifies the number of pyramid levels to build. Default
% is maxPyrHt(IM,FILT).  You can also specify 'auto' to use this value.
%
% FILT (optional) can be a string naming a standard filter (see
% namedFilter), or a vector which will be used for (separable)
% convolution.  Filter can be of even or odd length, but should be symmetric.
% Default = 'qmf9'.  EDGES specifies edge-handling, and
% defaults to 'reflect1' (see corrDn).
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.

% Eero Simoncelli, 6/96.

function [pyr,pind] = buildWpyr(im, ht, filt, edges)

if (nargin < 1)
  error('First argument (IM) is required');
end

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('filt') ~= 1)
  filt = 'qmf9';
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end

if isstr(filt)
  filt = namedFilter(filt);
end

if ( (size(filt,1) > 1) & (size(filt,2) > 1) )
  error('FILT should be a 1D filter (i.e., a vector)');
else
  filt = filt(:);
end

hfilt = modulateFlip(filt);

% Stagger sampling if filter is odd-length:
if (mod(size(filt,1),2) == 0)
  stag = 2;
else
  stag = 1;
end

im_sz = size(im);

max_ht = maxPyrHt(im_sz, size(filt,1));
if ( (exist('ht') ~= 1) | (ht == 'auto') )
  ht = max_ht;
else
  if (ht > max_ht)
    error(sprintf('Cannot build pyramid higher than %d levels.',max_ht));
  end
end

if (ht <= 0)

  pyr = im(:);
  pind = im_sz;

else

  if (im_sz(2) == 1)
    lolo = corrDn(im, filt, edges, [2 1], [stag 1]);
    hihi = corrDn(im, hfilt, edges, [2 1], [2 1]);
  elseif (im_sz(1) == 1)
    lolo = corrDn(im, filt', edges, [1 2], [1 stag]);
    hihi = corrDn(im, hfilt', edges, [1 2], [1 2]);
  else
    lo = corrDn(im, filt, edges, [2 1], [stag 1]);
    hi = corrDn(im, hfilt, edges, [2 1], [2 1]);
    lolo = corrDn(lo, filt', edges, [1 2], [1 stag]);
    lohi = corrDn(hi, filt', edges, [1 2], [1 stag]); % horizontal
    hilo = corrDn(lo, hfilt', edges, [1 2], [1 2]); % vertical
    hihi = corrDn(hi, hfilt', edges, [1 2], [1 2]); % diagonal
  end

  [npyr,nind] = buildWpyr(lolo, ht-1, filt, edges);

  if ((im_sz(1) == 1) | (im_sz(2) == 1))
    pyr = [hihi(:); npyr];
    pind = [size(hihi); nind];
  else
    pyr = [lohi(:); hilo(:); hihi(:); npyr];
    pind = [size(lohi); size(hilo); size(hihi); nind];
  end

end
```

Apr 26, 97 12:49 **cconv2.m** Page 1/1

```
% RES = CCONV2(MTX1, MTX2, CTR)
%
% Circular convolution of two matrices.  Result will be of size of
% LARGER vector.
%
% The origin of the smaller matrix is assumed to be its center.
% For even dimensions, the origin is determined by the CTR (optional)
% argument:
%      CTR   origin
%       0     DIM/2      (default)
%       1     (DIM/2)+1

% Eero Simoncelli, 6/96.  Modified 2/97.

function c = cconv2(a,b,ctr)

if (exist('ctr') ~= 1)
  ctr = 0;
end

if (( size(a,1) >= size(b,1) ) & ( size(a,2) >= size(b,2) ))
    large = a; small = b;
elseif  (( size(a,1) <= size(b,1) ) & ( size(a,2) <= size(b,2) ))
    large = b; small = a;
else
  error('one arg must be larger than the other in both dimensions!');
end

ly = size(large,1);
lx = size(large,2);
sy = size(small,1);
sx = size(small,2);

%% These values are the index of the small mtx that falls on the
%% border pixel of the large matrix when computing the first
%% convolution response sample:
sy2 = floor((sy+ctr+1)/2);
sx2 = floor((sx+ctr+1)/2);

% pad:
clarge = [ ...
    large(ly-sy+sy2+1:ly,lx-sx+sx2+1:lx), large(ly-sy+sy2+1:ly,:), ...
        large(ly-sy+sy2+1:ly,1:sx2-1); ...
    large(:,lx-sx+sx2+1:lx), large, large(:,1:sx2-1); ...
    large(1:sy2-1,lx-sx+sx2+1:lx), ...
        large(1:sy2-1,:), ...
        large(1:sy2-1,1:sx2-1) ];

c = conv2(clarge,small,'valid');
```

Oct 01, 02 17:54 **clip.m** Page 1/1

```
% [RES] = clip(IM, MINVALorRANGE, MAXVAL)
%
% Clip values of matrix IM to lie between minVal and maxVal:
%      RES = max(min(IM,MAXVAL),MINVAL)
% The first argument can also specify both min and max, as a 2-vector.
% If only one argument is passed, the range defaults to [0,1].

function res = clip(im, minValOrRange, maxVal)

if (exist('minValOrRange') ~= 1)
  minVal = 0;
  maxVal = 1;
elseif (length(minValOrRange) == 2)
  minVal = minValOrRange(1);
  maxVal = minValOrRange(2);
elseif (length(minValOrRange) == 1)
  minVal = minValOrRange;
  if (exist('maxVal') ~= 1)
    maxVal=minVal+1;
  end
else
  error('MINVAL must be  a scalar or a 2-vector');
end

if ( maxVal < minVal )
  error('MAXVAL should be less than MINVAL');
end

res = im;
res(find(im < minVal)) = minVal;
res(find(im > maxVal)) = maxVal;
```

Dec 16, 02 16:16 **columnize.m** Page 1/1

```
% [VEC] = columnize(MTX)
%
% Pack elements of MTX into a column vector.  Just provides a
% function-call notatoin for the operation MTX(:)

function vec = columnize(mtx)

vec = mtx(:);
```

```
% Image and Multi-scale Pyramid Tools
% Version 1.2,  June 2003.
% Created: Spring, 1996. Eero Simoncelli, eero.simoncelli@nyu.edu
%
% See README file for brief description.
% See ChangeLog file for latest modifications.
% See TUTORIALS subdirectory for demonstrations.
% Type "help <command-name>" for documentation on individual commands.
% -----------------------------------------------------------------
% Synthetic Images (matrices):
%   mkImpulse  - Make an image containing an impulse.
%   mkRamp     - Make an image containing a ramp function.
%   mkR        - Make an image containing distance from the origin.
%   mkAngle    - Make an image containing angle about origin.
%   mkDisc     - Make an image containing a disk image.
%   mkGaussian - Make an image containing a Gaussian function.
%   mkZonePlate - Make an image containing a zone plate (cos(r^2)).
%   mkAngularSine - Make an image containing an angular sine wave (pinwheel).
%   mkSine     - Make an image containing a sine grating.
%   mkSquare   - Make an image containing a square grating.
%   mkFract    - Make an image containing fractal (1/f) noise.
%
% Point Operations:
%   clip       - clip values to a range.
%   pointOp    - Lookup table (much faster than interp1) (MEX file).
%   histo      - Efficient histogram computation (MEX file).
%   histoMatch - Modify matrix elements to match specified histogram stats.
%
% Convolution (first two are significantly faster):
%   corrDn     - Correlate & downsample with boundary-handling (MEX file).
%   upConv     - Upsample & convolve with boundary-handling (MEX file).
%   blurDn     - Blur and subsample a signal/image.
%   upBlur     - Upsample and blur a signal/image.
%   cconv2     - Circular convolution.
%   rconv2     - Convolution with reflected boundaries.
%   zconv2     - Convolution assuming zeros beyond image boundaries.
%
% General pyramids:
%   pyrLow     - Access lowpass subband from (any type of) pyramid
%   pyrBand    - Access a subband from (any type of) pyramid
%   setPyrBand - Insert an image into (any type of) pyramid as a subband
%   pyrBandIndices - Returns indices for given band in a pyramid vector
%   maxPyrHt   - compute maximum number of scales in a pyramid
%
% Gaussian/Laplacian Pyramids:
%   buildGpyr  - Build a Gaussian pyramid of an input signal/image.
%   buildLpyr  - Build a Laplacian pyramid of an input signal/image.
%   reconLpyr  - Reconstruct (invert) the Laplacian pyramid transform.
%
% Separable orthonormal QMF/wavelet Pyramids:
%   buildWpyr  - Build a separable wavelet representation of an input signal/ima
ge.
%   reconWpyr  - Reconstruct (invert) the wavelet transform.
%   wpyrBand   - Extract a single band of the wavelet representation.
%   wpyrLev    - Extract (packed) subbands at a particular level
%   wpyrHt     - Number of levels (height) of a wavelet pyramid.
%
% Steerable Pyramids:
%   buildSpyr  - Build a steerable pyramid representation of an input image.
%   reconSpyr  - Reconstruct (invert) the steerable pyramid transform.
%   buildSFpyr - Build a steerable pyramid representation in the Fourier domain.
%   reconSFpyr - Reconstruct (invert) the (Fourier domain) steerable pyramid tra
nsform.
%   spyrBand   - Extract a single band from a steerable pyramid.
%   spyrHigh   - Highpass residual band.
%   spyrLev    - A whole level (i.e., all images at a given scale) of a steerabl
e pyramid.
%   spyrHt     - Number of levels (height) of a steerable pyramid.
%   spyrNumBands - Number of orientation bands in a steerable pyramid.
%
% Steerable filters:
%   steer      - Steer filters (or responses).
%   steer2HarmMtx - Construct a matrix mapping direcional basis to angular harmo
nics.
%
% Filters:
%   binomialFilter  - returns a filter of binomial coefficients.
%   namedFilter     - some typical Laplacian/Wavelet pyramid filters
%   spNFilters      - Set of Nth order steerable pyramid filters.
%   derivNFiltersS  - Matched set of S-tap 1D derivatives, orders 0 to N.
%
% Display:
%   showIm     - Display a matrix (real or complex) as grayscale image(s).
%                Displays dimensions, subsampling, and range of pixel values.
%   showLpyr   - Display a Laplacian pyramid.
%   showWpyr   - Display a separable wavelet pyramid.
%   showSpyr   - Display a steerable pyramid.
%   lplot      - "lollipop" plot.
%   nextFig    - Make next figure window current.
%   pixelAxes  - Make image display use an integer number of pixels
%                per sample to avoid resampling artifacts.
%
% Statistics (for 2D Matrices):
%   range2     - Min and max of image (matrix) (MEX file).
%   mean2      - Sample mean of an image (matrix).
%   var2       - Sample variance of an image (matrix).
%   skew2      - Sample skew (3rd moment / variance^1.5) of an image (matrix).
%   kurt2      - Sample kurtosis (4th moment / variance^2) of an image (matrix).

%   entropy2   - Sample entropy of an image (matrix).
%   imStats    - Report sample statistics of an image, or pair of images.
%
% Miscellaneous:
%   pgmRead    - Load a "pgm" image into a MatLab matrix.
%   pgmWrite   - Write a MatLab matrix to a "pgm" image file.
%   shift      - circular shift a 2D matrix by an arbitrary amount.
%   vectify    - pack matrix into column vector (i.e., function to compute mtx(:
)).
%   ifftshift  - inverse of MatLab's FFTSHIFT (differs for odd-length dimensions
)
%   rcosFn     - return a lookup table of a raised-cosine threshold fn.
%   innerProd  - Compute M'*M (M a matrix) efficiently (i.e., do not copy).
```



```
% RES = corrDn(IM, FILT, EDGES, STEP, START, STOP)
%
% Compute correlation of matrices IM with FILT, followed by
% downsampling.  These arguments should be 1D or 2D matrices, and IM
% must be larger (in both dimensions) than FILT.  The origin of filt
% is assumed to be floor(size(filt)/2)+1.
%
% EDGES is a string determining boundary handling:
%    'circular' - Circular convolution
%    'reflect1' - Reflect about the edge pixels
%    'reflect2' - Reflect, doubling the edge pixels
%    'repeat'   - Repeat the edge pixels
%    'zero'     - Assume values of zero outside image boundary
%    'extend'   - Reflect and invert
%    'dont-compute' - Zero output when filter overhangs input boundaries
%
% Downsampling factors are determined by STEP (optional, default=[1 1]),
% which should be a 2-vector [y,x].
%
% The window over which the convolution occurs is specfied by START
% (optional, default=[1,1], and STOP (optional, default=size(IM)).
%
% NOTE: this operation corresponds to multiplication of a signal
% vector by a matrix whose rows contain copies of the FILT shifted by
% multiples of STEP.  See upConv.m for the operation corresponding to
% the transpose of this matrix.

% Eero Simoncelli, 6/96, revised 2/97.

function res = corrDn(im, filt, edges, step, start, stop)

%% NOTE: THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

fprintf(1,'WARNING: You should compile the MEX version of "corrDn.c",\n         fo
und in the MEX subdirectory of matlabPyrTools, and put it in your matlab path.
It is MUCH faster, and provides more boundary-handling options.\n');

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('edges') == 1)
  if (strcmp(edges,'reflect1') ~= 1)
    warning('Using REFLECT1 edge-handling (use MEX code for other options).');
  end
end

if (exist('step') ~= 1)
        step = [1,1];
end

if (exist('start') ~= 1)
        start = [1,1];
end

if (exist('stop') ~= 1)
        stop = size(im);
end

%------------------------------------------------------------

% Reverse order of taps in filt, to do correlation instead of convolution
filt = filt(size(filt,1):-1:1,size(filt,2):-1:1);

tmp = rconv2(im,filt);
res = tmp(start(1):step(1):stop(1),start(2):step(2):stop(2));
```

Nov 30, 01 22:57 **entropy2.m** Page 1/1

```
% E = ENTROPY2(MTX,BINSIZE)
%
% Compute the first-order sample entropy of MTX.  Samples of VEC are
% first discretized.  Optional argument BINSIZE controls the
% discretization, and defaults to 256/(max(VEC)-min(VEC)).
%
% NOTE: This is a heavily  biased estimate of entropy when you
% don't have much data.

% Eero Simoncelli, 6/96.

function res = entropy2(mtx,binsize)

%% Ensure it's a vector, not a matrix.
vec = mtx(:);
[mn,mx] = range2(vec);

if (exist('binsize') == 1)
  nbins = max((mx-mn)/binsize, 1);
else
  nbins = 256;
end

[bincount,bins] = histo(vec,nbins);

%% Collect non-zero bins:
H = bincount(find(bincount));
H = H/sum(H);

res = -sum(H .* log2(H));
```

Dec 16, 02 16:19 **factorial.m** Page 1/1

```
%% RES = factorial(NUM)
%
% Factorial function that works on matrices (matlab's does not).

% EPS, 11/02

function res = factorial(num)

res = ones(size(num));

ind = find(num > 0);
if ( ~isempty(ind) )
  subNum = num(ind);
  res(ind) = subNum .* factorial(subNum-1);
end
```

```
% [N,X] = histo(MTX, nbinsOrBinsize, binCenter);
%
% Compute a histogram of (all) elements of MTX.  N contains the histogram
% counts, X is a vector containg the centers of the histogram bins.
%
% nbinsOrBinsize (optional, default = 101) specifies either
% the number of histogram bins, or the negative of the binsize.
%
% binCenter (optional, default = mean2(MTX)) specifies a center position
% for (any one of) the histogram bins.
%
% How does this differ from MatLab's HIST function?  This function:
%   - allows uniformly spaced bins only.
%   +/- operates on all elements of MTX, instead of columnwise.
%   + is much faster (approximately a factor of 80 on my machine).
%   + allows specification of number of bins OR binsize.  Default=101 bins.
%   + allows (optional) specification of binCenter.

% Eero Simoncelli, 3/97.

function [N, X] = histo(mtx, nbins, binCtr)

%% NOTE: THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

fprintf(1,'WARNING: You should compile the MEX version of "histo.c",\n         fou
nd in the MEX subdirectory of matlabPyrTools, and put it in your matlab path.  I
t is MUCH faster.\n');

mtx = mtx(:);

%------------------------------------------------------------
%% OPTIONAL ARGS:

[mn,mx] = range2(mtx);

if (exist('binCtr') ~= 1)
  binCtr =  mean(mtx);
end

if (exist('nbins') == 1)
  if (nbins < 0)
    binSize = -nbins;
  else
    binSize = ((mx-mn)/nbins);
    tmpNbins = round((mx-binCtr)/binSize) - round((mn-binCtr)/binSize);
    if (tmpNbins ~= nbins)
      warning('Using %d bins instead of requested number (%d)',tmpNbins,nbins);
    end
  end
else
  binSize = ((mx-mn)/101);
end

firstBin = binCtr + binSize*round( (mn-binCtr)/binSize );

tmpNbins = round((mx-binCtr)/binSize) - round((mn-binCtr)/binSize);

bins = firstBin + binSize*[0:tmpNbins];

[N, X] = hist(mtx, bins);
```



```
% RES = histoMatch(MTX, N, X)
%
% Modify elements of MTX so that normalized histogram matches that
% specified by vectors X and N, where N contains the histogram counts
% and X the histogram bin positions (see histo).

% Eero Simoncelli, 7/96.

function res = histoMatch(mtx, N, X)

if ( exist('histo') == 3 )
  [oN, oX] = histo(mtx(:), size(X(:),1));
else
  [oN, oX] = hist(mtx(:), size(X(:),1));
end

oStep = oX(2) - oX(1);
oC = [0, cumsum(oN)]/sum(oN);
oX = [oX(1)-oStep/2, oX+oStep/2];

N = N(:)';
X = X(:)';
N = N + mean(N)/(1e8);   %% HACK: no empty bins ensures nC strictly monotonic

nStep = X(2) - X(1);
nC = [0, cumsum(N)]/sum(N);
nX = [X(1)-nStep/2, X+nStep/2];

nnX = interp1(nC, nX, oC, 'linear');

if ( exist('pointOp') == 3 )
  res = pointOp(mtx, nnX, oX(1), oStep);
else
  res = reshape(interp1(oX, nnX, mtx(:)),size(mtx,1),size(mtx,2));
end
```

Apr 26, 97 12:49 **ifftshift.m** Page 1/1

```
% [RES] = ifftshift (MTX)
%
% Inverse of MatLab's FFTSHIFT.  That is,
%     ifftshift(fftshift(MTX)) = MTX
%   for any size MTX.

% Eero Simoncelli, 2/97.

function [res]  = ifftshift(mtx)

sz = size(mtx);
DC = ceil((sz+1)./2);                    % location of DC term in a matlab fft.

res = [mtx(DC(1):sz(1), DC(2):sz(2)) , mtx(DC(1):sz(1), 1:DC(2)-1); ...
       mtx(1:DC(1)-1, DC(2):sz(2)) , mtx(1:DC(1)-1, 1:DC(2)-1)];
```

Aug 21, 97 13:39 **imStats.m** Page 1/1

```
% imStats(IM1,IM2)
%
% Report image (matrix) statistics.
% When called on a single image IM1, report min, max, mean, stdev,
% and kurtosis.
% When called on two images (IM1 and IM2), report min, max, mean,
% stdev of the difference, and also SNR (relative to IM1).

% Eero Simoncelli, 6/96.

function [] = imStats(im1,im2)

if (~isreal(im1))
  error('Args must be real-valued matrices');
end

if (exist('im2') == 1)
  difference = im1 - im2;
  [mn,mx] = range2(difference);
  mean = mean2(difference);
  v = var2(difference,mean);
  if (v < realmin)
    snr = Inf;
  else
    snr = 10 * log10(var2(im1)/v);
  end
  fprintf(1, 'Difference statistics:\n');
  fprintf(1, '  Range: [%c, %c]\n',mn,mx);
  fprintf(1, '  Mean: %f,  Stdev (rmse): %f,  SNR (dB): %f\n',...
      mean,sqrt(v),snr);
else
  [mn,mx] = range2(im1);
  mean = mean2(im1);
  var = var2(im1);
  stdev = sqrt(real(var))+sqrt(imag(var));
  kurt = kurt2(im1, mean, stdev^2);
  fprintf(1, 'Image statistics:\n');
  fprintf(1, '  Range: [%f, %f]\n',mn,mx);
  fprintf(1, '  Mean: %f,  Stdev: %f,  Kurtosis: %f\n',mean,stdev,kurt);
end
```

```
% RES = innerProd(MTX)
%
% Compute (MTX' * MTX) efficiently (i.e., without copying the matrix)

function res = innerProd(mtx)

fprintf(1,['WARNING: You should compile the MEX version of' ...
           ' "innerProd.c",\n         found in the MEX subdirectory' ...
           ' of matlabPyrTools, and put it in your matlab path.' ...
           ' It is MUCH faster and requires less memory.\n']);

res = mtx' * mtx;
```



```
% K = KURT2(MTX,MEAN,VAR)
%
% Sample kurtosis (fourth moment divided by squared variance)
% of a matrix.  Kurtosis of a Gaussian distribution is 3.
%  MEAN (optional) and VAR (optional) make the computation faster.

% Eero Simoncelli, 6/96.

function res = kurt2(mtx, mn, v)

if (exist('mn') ~= 1)
        mn =  mean(mean(mtx));
end

if (exist('v') ~= 1)
        v =  var2(mtx,mn);
end

if (isreal(mtx))
  res = mean(mean(abs(mtx-mn).^4)) / (v^2);
else
  res = mean(mean(real(mtx-mn).^4)) / (real(v)^2) + ...
      i*mean(mean(imag(mtx-mn).^4)) / (imag(v)^2);
end
```

```
% lplot(VEC, XRANGE)
%
% Plot VEC, a vector, in  "lollipop" format.
% XRANGE (optional, default = [1,length(VEC)]), should be a 2-vector
% specifying the X positions (for labeling purposes) of the first and
% last sample of VEC.

% Mark Liberman, Linguistics Dept, UPenn, 1994.

function lplot(x,xrange)

if (exist('xrange') ~= 1)
  xrange = [1,length(x)];
end

msize = size(x);
if ( msize(2) == 1)
  x = x';
elseif (msize(1) ~= 1)
  error('First arg must be a vector');
end

if (~isreal(x))
  fprintf(1,'Warning: Imaginary part of signal ignored\n');
  x = abs(x);
end

N = length(x);
index = xrange(1) + (xrange(2)-xrange(1))*[0:(N-1)]/(N-1)
xinc = index(2)-index(1);

xx = [zeros(1,N);x;zeros(1,N)];
indexis = [index;index;index];
xdiscrete = [0 xx(:)' 0];
idiscrete = [index(1)-xinc indexis(:)' index(N)+xinc];

[mn,mx] = range2(xdiscrete);
ypad = (mx-mn)/12;                    % MAGIC NUMBER: graph padding

plot(idiscrete, xdiscrete, index, x, 'o');
axis([index(1)-xinc, index(N)+xinc, mn-ypad, mx+ypad]);

return
```



```
% [HEIGHT] = lpyrHt(INDICES)
%
% Compute height of Laplacian pyramid with given its INDICES matrix.
% See buildLpyr.m

% Eero Simoncelli, 6/96.

function [ht] =  lpyrHt(pind)

% Don't count lowpass residual band
ht = size(pind,1)-1;
```

Apr 26, 97 12:49 **maxPyrHt.m** Page 1/1

```
% HEIGHT = maxPyrHt(IMSIZE, FILTSIZE)
%
% Compute maximum pyramid height for given image and filter sizes.
% Specifically: the number of corrDn operations that can be sequentially
% performed when subsampling by a factor of 2.

% Eero Simoncelli, 6/96.

function height = maxPyrHt(imsz, filtsz)

imsz = imsz(:);
filtsz = filtsz(:);

if any(imsz == 1) % 1D image
  imsz = prod(imsz);
  filtsz = prod(filtsz);
elseif any(filtsz == 1)              % 2D image, 1D filter
  filtsz = [filtsz(1); filtsz(1)];
end

if any(imsz < filtsz)
  height = 0;
else
  height = 1 + maxPyrHt( floor(imsz/2), filtsz );
end
```

Apr 26, 97 12:40 **mean2.m** Page 1/1

```
% M = MEAN2(MTX)
%
% Sample mean of a matrix.

function res = mean2(mtx)

res = mean(mean(mtx));
```

```
% IM = mkAngle(SIZE, PHASE, ORIGIN)
%
% Compute a matrix of dimension SIZE (a [Y X] 2-vector, or a scalar)
% containing samples of the polar angle (in radians, CW from the
% X-axis, ranging from -pi to pi), relative to angle PHASE (default =
% 0), about ORIGIN pixel (default = (size+1)/2).

% Eero Simoncelli, 6/96.

function [res] = mkAngle(sz, phase, origin)

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz,sz];
end

% -----------------------------------------------------------------
% OPTIONAL args:

if (exist('origin') ~= 1)
  origin = (sz+1)/2;
end

% -----------------------------------------------------------------

[xramp,yramp] = meshgrid( [1:sz(2)]-origin(2), [1:sz(1)]-origin(1) );

res = atan2(yramp,xramp);

if (exist('phase') == 1)
  res = mod(res+(pi-phase),2*pi)-pi;
end
```



```
% IM = mkAngularSine(SIZE, HARMONIC, AMPL, PHASE, ORIGIN)
%
% Make an angular sinusoidal image:
%     AMPL * sin( HARMONIC*theta + PHASE),
% where theta is the angle about the origin.
% SIZE specifies the matrix size, as for zeros().
% AMPL (default = 1) and PHASE (default = 0) are optional.

% Eero Simoncelli, 2/97.

function [res] = mkAngularSine(sz, harmonic, ampl, ph, origin)

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz,sz];
end

mxsz = max(sz(1),sz(2));

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('harmonic') ~= 1)
  harmonic = 1;
end

if (exist('ampl') ~= 1)
  ampl = 1;
end

if (exist('ph') ~= 1)
  ph = 0;
end

if (exist('origin') ~= 1)
  origin = (sz+1)/2;
end

%------------------------------------------------------------

res = ampl * sin(harmonic*mkAngle(sz,ph,origin) + ph);
```

Apr 26, 97 12:49 **mkDisc.m** Page 1/1

```
% IM = mkDisc(SIZE, RADIUS, ORIGIN, TWIDTH, VALS)
%
% Make a "disk" image.  SIZE specifies the matrix size, as for
% zeros().  RADIUS (default = min(size)/4) specifies the radius of
% the disk.  ORIGIN (default = (size+1)/2) specifies the
% location of the disk center.  TWIDTH (in pixels, default = 2)
% specifies the width over which a soft threshold transition is made.
% VALS (default = [0,1]) should be a 2-vector containing the
% intensity value inside and outside the disk.

% Eero Simoncelli, 6/96.

function [res] = mkDisc(sz, rad, origin, twidth, vals)

if (nargin < 1)
  error('Must pass at least a size argument');
end

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz sz];
end

%------------------------------------------------------------
% OPTIONAL ARGS:

if (exist('rad') ~= 1)
  rad = min(sz(1),sz(2))/4;
end

if (exist('origin') ~= 1)
  origin = (sz+1)./2;
end

if (exist('twidth') ~= 1)
  twidth = 2;
end

if (exist('vals') ~= 1)
  vals = [1,0];
end

%------------------------------------------------------------

res = mkR(sz,1,origin);

if (abs(twidth) < realmin)
  res = vals(2) + (vals(1) - vals(2)) * (res <= rad);
else
  [Xtbl,Ytbl] = rcosFn(twidth, rad, [vals(1), vals(2)]);
  res = pointOp(res, Ytbl, Xtbl(1), Xtbl(2)-Xtbl(1), 0);
%
% OLD interp1 VERSION:
%  res = res(:);
%  Xtbl(1) = min(res);
%  Xtbl(size(Xtbl,2)) = max(res);
%  res = reshape(interp1(Xtbl,Ytbl,res), sz(1), sz(2));
%
end
```

Apr 26, 97 12:50 **mkFract.m** Page 1/1

```
% IM = mkFract(SIZE, FRACT_DIM)
%
% Make a matrix of dimensions SIZE (a [Y X] 2-vector, or a scalar)
% containing fractal (pink) noise with power spectral density of the
% form: 1/f^(5-2*FRACT_DIM).  Image variance is normalized to 1.0.
% FRACT_DIM defaults to 1.0

% Eero Simoncelli, 6/96.

%% TODO: Verify that this  matches Mandelbrot defn of fractal dimension.
%%       Make this more efficient!

function res = mkFract(dims, fract_dim)

if (exist('fract_dim') ~= 1)
  fract_dim = 1.0;
end

res = randn(dims);
fres = fft2(res);

sz = size(res);
ctr = ceil((sz+1)./2);

shape = ifftshift(mkR(sz, -(2.5-fract_dim), ctr));
shape(1,1) = 1;  %%DC term

fres = shape .* fres;
fres = ifft2(fres);

if (max(max(abs(imag(fres)))) > 1e-10)
  error('Symmetry error in creating fractal');
else
  res = real(fres);
  res = res / sqrt(var2(res));
end
```

```
% IM = mkGaussian(SIZE, COVARIANCE, MEAN, AMPLITUDE)
%
% Compute a matrix with dimensions SIZE (a [Y X] 2-vector, or a
% scalar) containing a Gaussian function, centered at pixel position
% specified by MEAN (default = (size+1)/2), with given COVARIANCE (can
% be a scalar, 2-vector, or 2x2 matrix.  Default = (min(size)/6)^2),
% and AMPLITUDE.  AMPLITUDE='norm' (default) will produce a
% probability-normalized function.  All but the first argument are
% optional.

% Eero Simoncelli, 6/96.

function [res] = mkGaussian(sz, cov, mn, ampl)

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz,sz];
end

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('cov') ~= 1)
  cov = (min(sz(1),sz(2))/6)^2;
end

if (exist('mn') ~= 1)
  mn = (sz+1)/2;
end

if (exist('ampl') ~= 1)
  ampl = 'norm';
end

%------------------------------------------------------------

[xramp,yramp] = meshgrid([1:sz(2)]-mn(2),[1:sz(1)]-mn(1));

if (sum(size(cov)) == 2)  % scalar
  if (strcmp(ampl,'norm'))
    ampl = 1/(2*pi*cov(1));
  end
  e = (xramp.^2 + yramp.^2)/(-2 * cov);
elseif (sum(size(cov)) == 3) % a 2-vector
  if (strcmp(ampl,'norm'))
    ampl = 1/(2*pi*sqrt(cov(1)*cov(2)));
  end
  e = xramp.^2/(-2 * cov(2)) + yramp.^2/(-2 * cov(1));
else
  if (strcmp(ampl,'norm'))
    ampl = 1/(2*pi*sqrt(det(cov)));
  end
  cov = -inv(cov)/2;
  e = cov(2,2)*xramp.^2 + (cov(1,2)+cov(2,1))*(xramp.*yramp) ...
      + cov(1,1)*yramp.^2;
end

res = ampl .* exp(e);
```



```
% IM = mkImpulse(SIZE, ORIGIN, AMPLITUDE)
%
% Compute a matrix of dimension SIZE (a [Y X] 2-vector, or a scalar)
% containing a single non-zero entry, at position ORIGIN (defaults to
% ceil(size/2)), of value AMPLITUDE (defaults to 1).

% Eero Simoncelli, 6/96.

function [res] = mkImpulse(sz, origin, amplitude)

sz = sz(:)';
if (size(sz,2) == 1)
  sz = [sz sz];
end

if (exist('origin') ~= 1)
  origin = ceil(sz/2);
end

if (exist('amplitude') ~= 1)
  amplitude = 1;
end

res = zeros(sz);
res(origin(1),origin(2)) = amplitude;
```

```
% IM = mkRamp(SIZE, DIRECTION, SLOPE, INTERCEPT, ORIGIN)
%
% Compute a matrix of dimension SIZE (a [Y X] 2-vector, or a scalar)
% containing samples of a ramp function, with given gradient DIRECTION
% (radians, CW from X-axis, default = 0), SLOPE (per pixel, default =
% 1), and a value of INTERCEPT (default = 0) at the ORIGIN (default =
% (size+1)/2, [1 1] = upper left).  All but the first argument are
% optional.

% Eero Simoncelli, 6/96. 2/97: adjusted coordinate system.

function [res] = mkRamp(sz, dir, slope, intercept, origin)

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz,sz];
end

% -----------------------------------------------------------------
% OPTIONAL args:

if (exist('dir') ~= 1)
  dir = 0;
end

if (exist('slope') ~= 1)
  slope = 1;
end

if (exist('intercept') ~= 1)
  intercept = 0;
end

if (exist('origin') ~= 1)
  origin = (sz+1)/2;
end

% -----------------------------------------------------------------

xinc = slope*cos(dir);
yinc = slope*sin(dir);

[xramp,yramp] = meshgrid( xinc*([1:sz(2)]-origin(2)), ...
    yinc*([1:sz(1)]-origin(1)) );

res = intercept + xramp + yramp;
```



```
% IM = mkR(SIZE, EXPT, ORIGIN)
%
% Compute a matrix of dimension SIZE (a [Y X] 2-vector, or a scalar)
% containing samples of a radial ramp function, raised to power EXPT
% (default = 1), with given ORIGIN (default = (size+1)/2, [1 1] =
% upper left).  All but the first argument are optional.

% Eero Simoncelli, 6/96.

function [res] = mkR(sz, expt, origin)

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz,sz];
end

% -----------------------------------------------------------------
% OPTIONAL args:

if (exist('expt') ~= 1)
  expt = 1;
end

if (exist('origin') ~= 1)
  origin = (sz+1)/2;
end

% -----------------------------------------------------------------

[xramp,yramp] = meshgrid( [1:sz(2)]-origin(2), [1:sz(1)]-origin(1) );

res = (xramp.^2 + yramp.^2).^(expt/2);
```

```
% IM = mkSine(SIZE, PERIOD, DIRECTION, AMPLITUDE, PHASE, ORIGIN)
%      or
% IM = mkSine(SIZE, FREQ, AMPLITUDE, PHASE, ORIGIN)
%
% Compute a matrix of dimension SIZE (a [Y X] 2-vector, or a scalar)
% containing samples of a 2D sinusoid, with given PERIOD (in pixels),
% DIRECTION (radians, CW from X-axis, default = 0), AMPLITUDE (default
% = 1), and PHASE (radians, relative to ORIGIN, default = 0).  ORIGIN
% defaults to the center of the image.
%
% In the second form, FREQ is a 2-vector of frequencies (radians/pixel).

% Eero Simoncelli, 6/96.

function [res] = mkSine(sz, per_freq, dir_amp, amp_phase, phase_orig, orig)

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (prod(size(per_freq)) == 2)
  frequency = norm(per_freq);
  direction = atan2(per_freq(1),per_freq(2));
  if (exist('dir_amp') == 1)
    amplitude = dir_amp;
  else
    amplitude = 1;
  end
  if (exist('amp_phase') == 1)
    phase = amp_phase;
  else
    phase = 0;
  end
  if (exist('phase_orig') == 1)
    origin = phase_orig;
  end
  if (exist('orig') == 1)
    error('Too many arguments for (second form) of mkSine');
  end
else
  frequency = 2*pi/per_freq;
  if (exist('dir_amp') == 1)
    direction = dir_amp;
  else
    direction = 0;
  end
  if (exist('amp_phase') == 1)
    amplitude = amp_phase;
  else
    amplitude = 1;
  end
  if (exist('phase_orig') == 1)
    phase = phase_orig;
  else
    phase = 0;
  end
  if (exist('orig') == 1)
    origin = orig;
  end
end

%------------------------------------------------------------

if (exist('origin') == 1)
  res = amplitude*sin(mkRamp(sz, direction, frequency, phase, origin));
else
 res = amplitude*sin(mkRamp(sz, direction, frequency, phase));
end
```



```
% IM = mkSquare(SIZE, PERIOD, DIRECTION, AMPLITUDE, PHASE, ORIGIN, TWIDTH)
%      or
% IM = mkSine(SIZE, FREQ, AMPLITUDE, PHASE, ORIGIN, TWIDTH)
%
% Compute a matrix of dimension SIZE (a [Y X] 2-vector, or a scalar)
% containing samples of a 2D square wave, with given PERIOD (in
% pixels), DIRECTION (radians, CW from X-axis, default = 0), AMPLITUDE
% (default = 1), and PHASE (radians, relative to ORIGIN, default = 0).
% ORIGIN defaults to the center of the image.  TWIDTH specifies width
% of raised-cosine edges on the bars of the grating (default =
% min(2,period/3)).
%
% In the second form, FREQ is a 2-vector of frequencies (radians/pixel).

% Eero Simoncelli, 6/96.

% TODO: Add duty cycle.

function [res] = mkSquare(sz, per_freq, dir_amp, amp_phase, phase_orig, orig_twi
dth, twidth)

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (prod(size(per_freq)) == 2)
  frequency = norm(per_freq);
  direction = atan2(per_freq(1),per_freq(2));
  if (exist('dir_amp') == 1)
    amplitude = dir_amp;
  else
    amplitude = 1;
  end
  if (exist('amp_phase') == 1)
    phase = amp_phase;
  else
    phase = 0;
  end
  if (exist('phase_orig') == 1)
    origin = phase_orig;
  end
  if (exist('orig_twidth') == 1)
    transition = orig_twidth;
  else
    transition = min(2,2*pi/(3*frequency));
  end
  if (exist('twidth') == 1)
    error('Too many arguments for (second form) of mkSine');
  end
else
  frequency = 2*pi/per_freq;
  if (exist('dir_amp') == 1)
    direction = dir_amp;
  else
    direction = 0;
  end
  if (exist('amp_phase') == 1)
    amplitude = amp_phase;
  else
    amplitude = 1;
  end
  if (exist('phase_orig') == 1)
    phase = phase_orig;
  else
    phase = 0;
  end
  if (exist('orig_twidth') == 1)
    origin = orig_twidth;
  end
  if (exist('twidth') == 1)
    transition = twidth;
  else
    transition = min(2,2*pi/(3*frequency));
  end

end

%------------------------------------------------------------

if (exist('origin') == 1)
  res = mkRamp(sz, direction, frequency, phase, origin) - pi/2;
else
  res = mkRamp(sz, direction, frequency, phase) - pi/2;
end

[Xtbl,Ytbl] = rcosFn(transition*frequency,pi/2,[-amplitude amplitude]);

res = pointOp(abs(mod(res+pi, 2*pi)-pi),Ytbl,Xtbl(1),Xtbl(2)-Xtbl(1),0);

% OLD threshold version:
%res = amplitude * (mod(res,2*pi) < pi);
```

```
% IM = mkZonePlate(SIZE, AMPL, PHASE)
%
% Make a "zone plate" image:
%     AMPL * cos( r^2 + PHASE)
% SIZE specifies the matrix size, as for zeros().
% AMPL (default = 1) and PHASE (default = 0) are optional.

% Eero Simoncelli, 6/96.

function [res] = mkZonePlate(sz, ampl, ph)

sz = sz(:);
if (size(sz,1) == 1)
  sz = [sz,sz];
end

mxsz = max(sz(1),sz(2));

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('ampl') ~= 1)
  ampl = 1;
end

if (exist('ph') ~= 1)
  ph = 0;
end

%------------------------------------------------------------

res = ampl * cos( (pi/mxsz) * mkR(sz,2) + ph );
```



```
% M = mod(A,B)
%
% Modulus operator: returns A mod B.
% Works on matrics, vectors or scalars.
%
% NOTE: This function is a Matlab-5 builtin, but was missing from Matlab-4.

% Eero Simoncelli, 7/96.

function m = mod(a,n)

m = a - n .* floor(a./n);
return;
```

Apr 26, 97 12:50 **modulateFlip.m** Page 1/1

```
% [HFILT] = modulateFlipShift(LFILT)
%
% QMF/Wavelet highpass filter construction: modulate by (-1)^n,
% reverse order (and shift by one, which is handled by the convolution
% routines).  This is an extension of the original definition of QMF's
% (e.g., see Simoncelli90).

% Eero Simoncelli, 7/96.

function [hfilt] = modulateFlipShift(lfilt)

lfilt = lfilt(:);

sz = size(lfilt,1);
sz2 = ceil(sz/2);

ind = [sz:-1:1]';

hfilt = lfilt(ind) .* (-1).^(ind-sz2);
```

Apr 26, 97 12:50 **namedFilter.m** Page 1/1

```
% KERNEL = NAMED_FILTER(NAME)
%
% Some standard 1D filter kernels.  These are scaled such that
% their L2-norm is 1.0.
%
%  binomN             - binomial coefficient filter of order N-1
%  haar:              - Haar wavelet.
%  qmf8, qmf12, qmf16 - Symmetric Quadrature Mirror Filters [Johnston80]
%  daub2,daub3,daub4  - Daubechies wavelet [Daubechies88].
%  qmf5, qmf9, qmf13: - Symmetric Quadrature Mirror Filters [Simoncelli88,Simonc
elli90]
%
%  See bottom of file for full citations.

% Eero Simoncelli, 6/96.

function [kernel] = named_filter(name)

if strcmp(name(1:min(5,size(name,2))), 'binom')
  kernel = sqrt(2) * binomialFilter(str2num(name(6:size(name,2))));
elseif strcmp(name,'qmf5')
  kernel = [-0.076103 0.3535534 0.8593118 0.3535534 -0.076103]';
elseif strcmp(name,'qmf9')
  kernel = [0.02807382 -0.060944743 -0.073386624 0.41472545 0.7973934 ...
      0.41472545 -0.073386624 -0.060944743 0.02807382]';
elseif strcmp(name,'qmf13')
  kernel = [-0.014556438 0.021651438 0.039045125 -0.09800052 ...
          -0.057827797 0.42995453 0.7737113 0.42995453 -0.057827797 ...
          -0.09800052 0.039045125 0.021651438 -0.014556438]';
elseif strcmp(name,'qmf8')
  kernel = sqrt(2) * [0.00938715 -0.07065183 0.06942827 0.4899808 ...
    0.4899808 0.06942827 -0.07065183 0.00938715 ]';
elseif strcmp(name,'qmf12')
  kernel = sqrt(2) * [-0.003809699 0.01885659 -0.002710326 -0.08469594 ...
        0.08846992 0.4843894 0.4843894 0.08846992 -0.08469594 -0.002710326 ...
        0.01885659 -0.003809699 ]';
elseif strcmp(name,'qmf16')
  kernel = sqrt(2) * [0.001050167 -0.005054526 -0.002589756 0.0276414 -0.0096663
76 ...
        -0.09039223 0.09779817 0.4810284 0.4810284 0.09779817 -0.09039223 -0.009
666376 ...
        0.0276414 -0.002589756 -0.005054526 0.001050167 ]';
elseif strcmp(name,'haar')
  kernel = [1 1]' / sqrt(2);
elseif strcmp(name,'daub2')
  kernel = [0.482962913145 0.836516303738 0.224143868042 -0.129409522551]';
elseif strcmp(name,'daub3')
  kernel = [0.332670552950 0.806891509311 0.459877502118 -0.135011020010 ...
        -0.085441273882  0.035226291882]';
elseif strcmp(name,'daub4')
  kernel = [0.230377813309 0.714846570553 0.630880767930 -0.027983769417 ...
        -0.187034811719 0.030841381836 0.032883011667 -0.010597401785]';
elseif strcmp(name,'gauss5')  % for backward-compatibility
  kernel = sqrt(2) * [0.0625 0.25 0.375 0.25 0.0625]';
elseif strcmp(name,'gauss3')  % for backward-compatibility
  kernel = sqrt(2) * [0.25 0.5 0.25]';
else
  error(sprintf('Bad filter name: %s\n',name));
end

% [Johnston80] - J D Johnston, "A filter family designed for use in quadrature
%   mirror filter banks", Proc. ICASSP, pp 291-294, 1980.
%
% [Daubechies88] - I Daubechies, "Orthonormal bases of compactly supported wavelets",
%    Commun. Pure Appl. Math, vol. 42, pp 909-996, 1988.
%
% [Simoncelli88] - E P Simoncelli,  "Orthogonal sub-band image transforms",
%     PhD Thesis, MIT Dept. of Elec. Eng. and Comp. Sci. May 1988.
%     Also available as: MIT Media Laboratory Vision and Modeling Technical
%     Report #100.
%
% [Simoncelli90] -  E P Simoncelli and E H Adelson, "Subband image coding",
%    Subband Transforms, chapter 4, ed. John W Woods, Kluwer Academic
%    Publishers,  Norwell, MA, 1990, pp 143--192.
```

Apr 26, 97 12:50 **nextFig.m** Page 1/1

```
% nextFig (MAXFIGS, SKIP)
%
% Make figure number mod((GCF+SKIP), MAXFIGS) the current figure.
% MAXFIGS is optional, and defaults to 2.
% SKIP is optional, and defaults to 1.

% Eero Simoncelli, 2/97.

function nextFig(maxfigs, skip)

if (exist('maxfigs') ~= 1)
  maxfigs = 2;
end

if (exist('skip') ~= 1)
  skip = 1;
end

figure(1+mod(gcf-1+skip,maxfigs));
```

May 10, 97 14:43 **pgmRead.m** Page 1/1

```
% IM = pgmRead( FILENAME )
%
% Load a pgm image into a MatLab matrix.
%   This format is accessible from the XV image browsing utility.
%   Only works for 8bit gray images (raw or ascii)

% Hany Farid, Spring '96.  Modified by Eero Simoncelli, 6/96.

function im = pgmRead( fname );

[fid,msg] = fopen( fname, 'r' );

if (fid == -1)
  error(msg);
end

%%% First line contains ID string:
%%% "P1" = ascii bitmap, "P2" = ascii greymap,
%%% "P3" = ascii pixmap, "P4" = raw bitmap,
%%% "P5" = raw greymap, "P6" = raw pixmap
TheLine = fgetl(fid);
format  = TheLine;

if ~((format(1:2) == 'P2') | (format(1:2) == 'P5'))
  error('PGM file must be of type P2 or P5');
end

%%% Any number of comment lines
TheLine  = fgetl(fid);
while TheLine(1) == '#'
        TheLine = fgetl(fid);
end

%%% dimensions
sz = sscanf(TheLine,'%d',2);
xdim = sz(1);
ydim = sz(2);
sz = xdim * ydim;

%%% Maximum pixel value
TheLine  = fgetl(fid);
maxval = sscanf(TheLine, '%d',1);

%%im  = zeros(dim,1);
if (format(2) == '2')
  [im,count]  = fscanf(fid,'%d',sz);
else
  [im,count]  = fread(fid,sz,'uchar');
end

fclose(fid);

if (count == sz)
  im = reshape( im, xdim, ydim )';
else
  fprintf(1,'Warning: File ended early!');
  im = reshape( [im ; zeros(sz-count,1)], xdim, ydim)';
end
```

Apr 16, 98 18:10 **pgmWrite.m** Page 1/1

```
% RANGE = pgmWrite(MTX, FILENAME, RANGE, TYPE, COMMENT)
%
% Write a MatLab matrix to a pgm (graylevel image) file.
% This format is accessible from the XV image browsing utility.
%
% RANGE (optional) is a 2-vector specifying the values that map to
% black and white, respectively.  Passing a value of 'auto' (default)
% sets RANGE=[min,max] (as in MatLab's imagesc).  'auto2' sets
% RANGE=[mean-2*stdev, mean+2*stdev].  'auto3' sets
% RANGE=[p1-(p2-p1)/8, p2+(p2-p1)/8], where p1 is the 10th percentile
% value of the sorted MATRIX samples, and p2 is the 90th percentile
% value.
%
% TYPE (optional) should be 'raw' or 'ascii'.  Defaults to 'raw'.

% Hany Farid,  Spring '96.  Modified by Eero Simoncelli, 6/96.

function range = pgmWrite(mtx, fname, range, type, comment );

[fid,msg] = fopen( fname, 'w' );

if (fid == -1)
  error(msg);
end

%-------------------------------------------------------------
%% optional ARGS:

if (exist('range') ~= 1)
  range = 'auto';
end

if (exist('type') ~= 1)
  type = 'raw';
end
%-------------------------------------------------------------

%% Automatic range calculation:
if (strcmp(range,'auto1') | strcmp(range,'auto'))
  [mn,mx] = range2(mtx);
  range = [mn,mx];

elseif strcmp(range,'auto2')
  stdev = sqrt(var2(mtx));
  av = mean2(mtx);
  range = [av-2*stdev,av+2*stdev];      % MAGIC NUMBER: 2 stdevs

elseif strcmp(range, 'auto3')
  percentile = 0.1;                     % MAGIC NUMBER: 0<p<0.5
  [N,X] = histo(mtx);
  binsz = X(2)-X(1);
  N = N+1e-10;  % Ensure cumsum will be monotonic for call to interp1
  cumN = [0, cumsum(N)]/sum(N);
  cumX = [X(1)-binsz, X] + (binsz/2);
  ctrRange = interp1(cumN,cumX, [percentile, 1-percentile]);
  range = mean(ctrRange) + (ctrRange-mean(ctrRange))/(1-2*percentile);

elseif isstr(range)
  error(sprintf('Bad RANGE argument: %s',range))

end

if ((range(2) - range(1)) <= eps)
  range(1) = range(1) - 0.5;
  range(2) = range(2) + 0.5;
end


%%% First line contains ID string:
%%% "P1" = ascii bitmap, "P2" = ascii greymap,
%%% "P3" = ascii pixmap, "P4" = raw bitmap,
%%% "P5" = raw greymap, "P6" = raw pixmap
if strcmp(type,'raw')
  fprintf(fid,'P5\n');
  format = 5;
elseif strcmp(type,'ascii')
  fprintf(fid,'P2\n');
  format = 2;
else
  error(sprintf('PGMWRITE: Bad type argument: %s',type));
end

fprintf(fid,'# MatLab PGMWRITE file, saved %s\n',date);

if (exist('comment') == 1)
  fprintf(fid,'# %s\n', comment);
end

%%% dimensions
fprintf(fid,'%d %d\n',size(mtx,2),size(mtx,1));

%%% Maximum pixel value
fprintf(fid,'255\n');


%% MatLab's "fprintf" floors when writing floats, so we compute
%% (mtx-r1)*255/(r2-r1)+0.5
mult = (255 / (range(2)-range(1)));
mtx = (mult * mtx) + (0.5 - mult * range(1));

mtx = max(-0.5+eps,min(255.5-eps,mtx));

if (format == 2)
  count  = fprintf(fid,'%d ',mtx);
elseif (format == 5)
  count  = fwrite(fid,mtx','uchar');
end

fclose(fid);

if (count ~= size(mtx,1)*size(mtx,2))
  fprintf(1,'Warning: File output terminated early!');
end

%%% TEST:
% foo = 257*rand(100)-1;
% pgmWrite(foo,'foo.pgm',[0 255]);
% foo2=pgmRead('foo.pgm');
% size(find((foo2-round(foo))~=0))
% foo(find((foo2-round(foo))~=0))
```

Feb 17, 98 15:01 **pixelAxes.m** Page 1/1

```
% [ZOOM] = pixelAxes(DIMS, ZOOM)
%
% Set the axes of the current plot to cover a multiple of DIMS pixels,
% thereby eliminating screen aliasing artifacts when displaying an
% image of size DIMS.
%
% ZOOM (optional, default='same') expresses the desired number of
% samples displayed per screen pixel.  It should be a scalar, which
% will be rounded to the nearest integer, or 1 over an integer.  It
% may also be the string 'same' or 'auto', in which case the value is chosen so
% as to produce an image closest in size to the currently displayed
% image.  It may also be the string 'full', in which case the image is
% made as large as possible while still fitting in the window.

% Eero Simoncelli, 2/97.

function [zoom] = pixelAxes(dims, zoom)

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('zoom') ~= 1)
  zoom = 'same';
end

%% Reverse  dimension order, since Figure Positions reported as (x,y).
dims = dims(2:-1:1);

%% Use MatLab's axis function to force square pixels, etc:
axis('image');
ax = gca;

oldunits = get(ax,'Units');

if strcmp(zoom,'full');
  set(ax,'Units','normalized');
  set(ax,'Position',[0 0 1 1]);
  zoom = 'same';
end

set(ax,'Units','pixels');
pos = get(ax,'Position');
ctr = pos(1:2)+pos(3:4)/2;

if (strcmp(zoom,'same') | strcmp(zoom,'auto'))
  %% HACK: enlarge slightly so that floor doesn't round down
  zoom = min( pos(3:4) ./ (dims - 1) );
elseif isstr(zoom)
  error(sprintf('Bad ZOOM argument: %s',zoom));
end

%% Force zoom value to be an integer, or inverse integer.
if (zoom < 0.75)
  zoom = 1/ceil(1/zoom);
  %% Round upward, subtracting 0.5 to avoid floating point errors.
  newsz = ceil(zoom*(dims-0.5));
else
  zoom = floor(zoom + 0.001);           % Avoid floating pt errors
  if (zoom < 1.5)                       % zoom=1
    zoom = 1;
    newsz = dims + 0.5;
  else
    newsz = zoom*(dims-1) + mod(zoom,2);
  end
end

set(ax,'Position', [floor(ctr-newsz/2)+0.5, newsz] )

% Restore units
set(ax,'Units',oldunits);
```

Sep 18, 02 18:09 **pointOp.m** Page 1/1

```
% RES = pointOp(IM, LUT, ORIGIN, INCREMENT, WARNINGS)
%
% Apply a point operation, specified by lookup table LUT, to image IM.
% LUT must be a row or column vector, and is assumed to contain
% (equi-spaced) samples of the function.  ORIGIN specifies the
% abscissa associated with the first sample, and INCREMENT specifies the
% spacing between samples.  Between-sample values are estimated via
% linear interpolation.  If WARNINGS is non-zero, the function prints
% a warning whenever the lookup table is extrapolated.
%
% This function is much faster than MatLab's interp1, and allows
% extrapolation beyond the lookup table domain.  The drawbacks are
% that the lookup table must be equi-spaced, and the interpolation is
% linear.

% Eero Simoncelli, 8/96.

function res = pointOp(im, lut, origin, increment, warnings)

%% NOTE: THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

fprintf(1,'WARNING: You should compile the MEX version of "pointOp.c",\n         f
ound in the MEX subdirectory of matlabPyrTools, and put it in your matlab path.
 It is MUCH faster.\n');

X = origin + increment*[0:size(lut(:),1)-1];
Y = lut(:);

res = reshape(interp1(X, Y, im(:), 'linear', 'extrap'),size(im));
```

Jul 17, 96 1:30 **pwd2path.m** Page 1/1

```
% PWD2PATH()
%
% add current working directory (pwd) to path.

P = path;
path(pwd,P);
```

```
% RES = pyrBandIndices(INDICES, BAND_NUM)
%
% Return indices for accessing a subband from a pyramid
% (gaussian, laplacian, QMF/wavelet, steerable).

% Eero Simoncelli, 6/96.

function indices =  pyrBandIndices(pind,band)

if ((band > size(pind,1)) | (band < 1))
  error(sprintf('BAND_NUM must be between 1 and number of pyramid bands (%d).',
...
      size(pind,1)));
end

if (size(pind,2) ~= 2)
  error('INDICES must be an Nx2 matrix indicating the size of the pyramid subban
ds');
end

ind = 1;
for l=1:band-1
  ind = ind + prod(pind(l,:));
end

indices = ind:ind+prod(pind(band,:))-1;
```



```
% RES = pyrBand(PYR, INDICES, BAND_NUM)
%
% Access a subband from a pyramid (gaussian, laplacian, QMF/wavelet,
% or steerable).  Subbands are numbered consecutively, from finest
% (highest spatial frequency) to coarsest (lowest spatial frequency).

% Eero Simoncelli, 6/96.

function res =  pyrBand(pyr, pind, band)

res = reshape( pyr(pyrBandIndices(pind,band)), pind(band,1), pind(band,2) );
```

```
% RES = pyrLow(PYR, INDICES)
%
% Access the lowpass subband from a pyramid
%   (gaussian, laplacian, QMF/wavelet, steerable).

% Eero Simoncelli, 6/96.

function res =  pyrLow(pyr,pind)

band = size(pind,1);

res = reshape( pyr(pyrBandIndices(pind,band)), pind(band,1), pind(band,2) );
```



```
% [MIN, MAX] = range2(MTX)
%
% Compute minimum and maximum values of MTX, returning them as a 2-vector.

% Eero Simoncelli, 3/97.

function [mn, mx] = range2(mtx)

%% NOTE: THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

fprintf(1,'WARNING: You should compile the MEX version of "range2.c",\n         fo
und in the MEX subdirectory of matlabPyrTools, and put it in your matlab path.
It is MUCH faster.\n');

if (~isreal(mtx))
  error('MTX must be real-valued');
end

mn = min(min(mtx));
mx = max(max(mtx));
```

```
% RES = RCONV2(MTX1, MTX2, CTR)
%
% Convolution of two matrices, with boundaries handled via reflection
% about the edge pixels.  Result will be of size of LARGER matrix.
%
% The origin of the smaller matrix is assumed to be its center.
% For even dimensions, the origin is determined by the CTR (optional)
% argument:
%      CTR   origin
%       0     DIM/2      (default)
%       1     (DIM/2)+1

% Eero Simoncelli, 6/96.

function c = rconv2(a,b,ctr)

if (exist('ctr') ~= 1)
  ctr = 0;
end

if (( size(a,1) >= size(b,1) ) & ( size(a,2) >= size(b,2) ))
    large = a; small = b;
elseif  (( size(a,1) <= size(b,1) ) & ( size(a,2) <= size(b,2) ))
    large = b; small = a;
else
  error('one arg must be larger than the other in both dimensions!');
end

ly = size(large,1);
lx = size(large,2);
sy = size(small,1);
sx = size(small,2);

%% These values are one less than the index of the small mtx that falls on
%% the border pixel of the large matrix when computing the first
%% convolution response sample:
sy2 = floor((sy+ctr-1)/2);
sx2 = floor((sx+ctr-1)/2);

% pad with reflected copies
clarge = [
    large(sy-sy2:-1:2,sx-sx2:-1:2), large(sy-sy2:-1:2,:), ...
        large(sy-sy2:-1:2,lx-1:-1:lx-sx2); ...
    large(:,sx-sx2:-1:2),    large,   large(:,lx-1:-1:lx-sx2); ...
    large(ly-1:-1:ly-sy2,sx-sx2:-1:2), ...
      large(ly-1:-1:ly-sy2,:), ...
      large(ly-1:-1:ly-sy2,lx-1:-1:lx-sx2) ];

c = conv2(clarge,small,'valid');
```



```
% [X, Y] = rcosFn(WIDTH, POSITION, VALUES)
%
% Return a lookup table (suitable for use by INTERP1)
% containing a "raised cosine" soft threshold function:
%
%    Y =  VALUES(1) + (VALUES(2)-VALUES(1)) *
%              cos^2( PI/2 * (X - POSITION + WIDTH)/WIDTH )
%
% WIDTH is the width of the region over which the transition occurs
% (default = 1). POSITION is the location of the center of the
% threshold (default = 0).  VALUES (default = [0,1]) specifies the
% values to the left and right of the transition.

% Eero Simoncelli, 7/96.

function [X, Y] = rcosFn(width,position,values)

%------------------------------------------------------------
% OPTIONAL ARGS:

if (exist('width') ~= 1)
  width = 1;
end

if (exist('position') ~= 1)
  position = 0;
end

if (exist('values') ~= 1)
  values = [0,1];
end

%------------------------------------------------------------

sz = 256;  %% arbitrary!

X    = pi * [-sz-1:1] / (2*sz);

Y = values(1) + (values(2)-values(1)) * cos(X).^2;

%    Make sure end values are repeated, for extrapolation...
Y(1) = Y(2);
Y(sz+3) = Y(sz+2);

X = position + (2*width/pi) * (X + pi/4);
```

```
% RES = reconLpyr(PYR, INDICES, LEVS, FILT2, EDGES)
%
% Reconstruct image from Laplacian pyramid, as created by buildLpyr.
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.
%
% LEVS (optional) should be a list of levels to include, or the string
% 'all' (default).  The finest scale is number 1.  The lowpass band
% corresponds to lpyrHt(INDICES)+1.
%
% FILT2 (optional) can be a string naming a standard filter (see
% namedFilter), or a vector which will be used for (separable)
% convolution.  Default = 'binom5'.  EDGES specifies edge-handling,
% and defaults to 'reflect1' (see corrDn).

% Eero Simoncelli, 6/96

function res = reconLpyr(pyr, ind, levs, filt2, edges)

if (nargin < 2)
  error('First two arguments (PYR, INDICES) are required');
end

%%------------------------------------------------------------
%% DEFAULTS:

if (exist('levs') ~= 1)
  levs = 'all';
end

if (exist('filt2') ~= 1)
  filt2 = 'binom5';
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end
%%------------------------------------------------------------

maxLev =  1+lpyrHt(ind);
if strcmp(levs,'all')
  levs = [1:maxLev]';
else
  if (any(levs > maxLev))
    error(sprintf('Level numbers must be in the range [1, %d].', maxLev));
  end
  levs = levs(:);
end

if isstr(filt2)
  filt2 = namedFilter(filt2);
end

filt2 = filt2(:);
res_sz = ind(1,:);

if any(levs > 1)

  int_sz = [ind(1,1), ind(2,2)];

  nres = reconLpyr( pyr(prod(res_sz)+1:size(pyr,1)), ...
      ind(2:size(ind,1),:), levs-1, filt2, edges);

  if (res_sz(1) == 1)
    res = upConv(nres, filt2', edges, [1 2], [1 1], res_sz);
  elseif (res_sz(2) == 1)
    res = upConv(nres, filt2, edges, [2 1], [1 1], res_sz);
  else
    hi = upConv(nres, filt2, edges, [2 1], [1 1], int_sz);
    res = upConv(hi, filt2', edges, [1 2], [1 1], res_sz);
  end

else

  res = zeros(res_sz);

end

if any(levs == 1)
  res = res + pyrBand(pyr,ind,1);
end
```



```
% RESDFT = reconSFpyrLevs(PYR,INDICES,LOGRAD,XRCOS,YRCOS,ANGLE,NBANDS,LEVS,BANDS
)
%
% Recursive function for reconstructing levels of a steerable pyramid
% representation.  This is called by reconSFpyr, and is not usually
% called directly.

% Eero Simoncelli, 5/97.

function resdft = reconSFpyrLevs(pyr,pind,log_rad,Xrcos,Yrcos,angle,nbands,levs,
bands);

lo_ind = nbands+1;
dims = pind(1,:);
ctr = ceil((dims+0.5)/2);

%  log_rad = log_rad + 1;
Xrcos = Xrcos - log2(2);  % shift origin of lut by 1 octave.

if any(levs > 1)

  lodims = ceil((dims-0.5)/2);
  loctr = ceil((lodims+0.5)/2);
  lostart = ctr-loctr+1;
  loend = lostart+lodims-1;
  nlog_rad = log_rad(lostart(1):loend(1),lostart(2):loend(2));
  nangle = angle(lostart(1):loend(1),lostart(2):loend(2));

  if  (size(pind,1) > lo_ind)
    nresdft = reconSFpyrLevs( pyr(1+sum(prod(pind(1:lo_ind-1,:)')):size(pyr,1)),
...
        pind(lo_ind:size(pind,1),:), ...
        nlog_rad, Xrcos, Yrcos, nangle, nbands,levs-1, bands);
  else
    nresdft = fftshift(fft2(pyrBand(pyr,pind,lo_ind)));
  end

  YIrcos = sqrt(abs(1.0 - Yrcos.^2));
  lomask = pointOp(nlog_rad, YIrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);

  resdft = zeros(dims);
  resdft(lostart(1):loend(1),lostart(2):loend(2)) = nresdft .* lomask;

else

  resdft = zeros(dims);

end


if any(levs == 1)

  lutsize = 1024;
  Xcosn = pi*[-(2*lutsize+1):(lutsize+1)]/lutsize;  % [-2*pi:pi]
  order = nbands-1;
  %% divide by sqrt(sum_(n=0)^(N-1)  cos(pi*n/N)^(2(N-1)) )
  const = (2^(2*order))*(factorial(order)^2)/(nbands*factorial(2*order));
  Ycosn = sqrt(const) * (cos(Xcosn)).^order;
  himask = pointOp(log_rad, Yrcos, Xrcos(1), Xrcos(2)-Xrcos(1),0);

  ind = 1;
  for b = 1:nbands
    if any(bands == b)
      anglemask = pointOp(angle,Ycosn,Xcosn(1)+pi*(b-1)/nbands,Xcosn(2)-Xcosn(1)
);
      band = reshape(pyr(ind:ind+prod(dims)-1), dims(1), dims(2));
      banddft = fftshift(fft2(band));
      resdft = resdft + (sqrt(-1))^(nbands-1) * banddft.*anglemask.*himask;
    end
    ind = ind + prod(dims);
  end
end
```

```
% RES = reconSFpyr(PYR, INDICES, LEVS, BANDS, TWIDTH)
%
% Reconstruct image from its steerable pyramid representation, in the Fourier
% domain, as created by buildSFpyr.
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.
%
% LEVS (optional) should be a list of levels to include, or the string
% 'all' (default).  0 corresonds to the residual highpass subband.
% 1 corresponds to the finest oriented scale.  The lowpass band
% corresponds to number spyrHt(INDICES)+1.
%
% BANDS (optional) should be a list of bands to include, or the string
% 'all' (default).  1 = vertical, rest proceeding anti-clockwise.
%
% TWIDTH is the width of the transition region of the radial lowpass
% function, in octaves (default = 1, which gives a raised cosine for
% the bandpass filters).

% Eero Simoncelli, 5/97.

function res = reconSFpyr(pyr, pind, levs, bands, twidth)

%%------------------------------------------------------------
%% DEFAULTS:

if (exist('levs') ~= 1)
  levs = 'all';
end

if (exist('bands') ~= 1)
  bands = 'all';
end

if (exist('twidth') ~= 1)
  twidth = 1;
elseif (twidth <= 0)
  fprintf(1,'Warning: TWIDTH must be positive.  Setting to 1.\n');
  twidth = 1;
end

%%------------------------------------------------------------

nbands = spyrNumBands(pind);

maxLev =  1+spyrHt(pind);
if strcmp(levs,'all')
  levs = [0:maxLev]';
else
  if (any(levs > maxLev) | any(levs < 0))
    error(sprintf('Level numbers must be in the range [0, %d].', maxLev));
  end
  levs = levs(:);
end

if strcmp(bands,'all')
  bands = [1:nbands]';
else
  if (any(bands < 1) | any(bands > nbands))
    error(sprintf('Band numbers must be in the range [1,3].', nbands));
  end
  bands = bands(:);
end

%----------------------------------------------------------------------

dims = pind(1,:);
ctr = ceil((dims+0.5)/2);

[xramp,yramp] = meshgrid( ([1:dims(2)]-ctr(2))./(dims(2)/2), ...
    ([1:dims(1)]-ctr(1))./(dims(1)/2) );
angle = atan2(yramp,xramp);
log_rad = sqrt(xramp.^2 + yramp.^2);
log_rad(ctr(1),ctr(2)) =  log_rad(ctr(1),ctr(2)-1);
log_rad  = log2(log_rad);

%% Radial transition function (a raised cosine in log-frequency):
[Xrcos,Yrcos] = rcosFn(twidth,(-twidth/2),[0 1]);
Yrcos = sqrt(Yrcos);
YIrcos = sqrt(abs(1.0 - Yrcos.^2));

if (size(pind,1) == 2)
  if (any(levs==1))
    resdft = fftshift(fft2(pyrBand(pyr,pind,2)));
  else
    resdft = zeros(pind(2,:));
  end
else
  resdft = reconSFpyrLevs(pyr(1+prod(pind(1,:)):size(pyr,1)), ...
      pind(2:size(pind,1),:), ...
      log_rad, Xrcos, Yrcos, angle, nbands, levs, bands);
end

lo0mask = pointOp(log_rad, YIrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);
resdft = resdft .* lo0mask;

%% residual highpass subband
if any(levs == 0)
  hi0mask = pointOp(log_rad, Yrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);
  hidft = fftshift(fft2(subMtx(pyr, pind(1,:))));
  resdft = resdft + hidft .* hi0mask;
end

res = real(ifft2(ifftshift(resdft)));
```



```
% RES = reconSpyrLevs(PYR,INDICES,LOFILT,BFILTS,EDGES,LEVS,BANDS)
%
% Recursive function for reconstructing levels of a steerable pyramid
% representation.  This is called by reconSpyr, and is not usually
% called directly.

% Eero Simoncelli, 6/96.

function res = reconSpyrLevs(pyr,pind,lofilt,bfilts,edges,levs,bands);

nbands = size(bfilts,2);
lo_ind = nbands+1;
res_sz = pind(1,:);

% Assume square filters:
bfiltsz =  round(sqrt(size(bfilts,1)));

if any(levs > 1)

  if  (size(pind,1) > lo_ind)
    nres = reconSpyrLevs( pyr(1+sum(prod(pind(1:lo_ind-1,:)')):size(pyr,1)),  ..
.
        pind(lo_ind:size(pind,1),:), ...
        lofilt, bfilts, edges, levs-1, bands);
  else
    nres = pyrBand(pyr,pind,lo_ind);    % lowpass subband
  end

  res = upConv(nres, lofilt, edges, [2 2], [1 1], res_sz);

else

  res = zeros(res_sz);

end

if any(levs == 1)
  ind = 1;
  for b = 1:nbands
    if any(bands == b)
      bfilt = reshape(bfilts(:,b), bfiltsz, bfiltsz);
      upConv(reshape(pyr(ind:ind+prod(res_sz)-1), res_sz(1), res_sz(2)), ...
             bfilt, edges, [1 1], [1 1], res_sz, res);
    end
    ind = ind + prod(res_sz);
  end
end
```

```
% RES = reconSpyr(PYR, INDICES, FILTFILE, EDGES, LEVS, BANDS)
%
% Reconstruct image from its steerable pyramid representation, as created
% by buildSpyr.
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.
%
% FILTFILE (optional) should be a string referring to an m-file that returns
% the rfilters.  examples: sp0Filters, sp1Filters, sp3Filters
% (default = 'sp1Filters').
% EDGES specifies edge-handling, and defaults to 'reflect1' (see
% corrDn).
%
% LEVS (optional) should be a list of levels to include, or the string
% 'all' (default).  0 corresonds to the residual highpass subband.
% 1 corresponds to the finest oriented scale.  The lowpass band
% corresponds to number spyrHt(INDICES)+1.
%
% BANDS (optional) should be a list of bands to include, or the string
% 'all' (default).  1 = vertical, rest proceeding anti-clockwise.

% Eero Simoncelli, 6/96.

function res = reconSpyr(pyr, pind, filtfile, edges, levs, bands)

%%------------------------------------------------------------
%% DEFAULTS:

if (exist('filtfile') ~= 1)
  filtfile = 'sp1Filters';
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end

if (exist('levs') ~= 1)
  levs = 'all';
end

if (exist('bands') ~= 1)
  bands = 'all';
end

%%------------------------------------------------------------

if (isstr(filtfile) & (exist(filtfile) == 2))
   [lo0filt,hi0filt,lofilt,bfilts,steermtx,harmonics] = eval(filtfile);
   nbands = spyrNumBands(pind);
   if ((nbands > 0) & (size(bfilts,2) ~= nbands))
     error('Number of pyramid bands is inconsistent with filter file');
   end
else
  error('filtfile argument must be the name of an M-file containing SPYR filters
.');
end

maxLev =  1+spyrHt(pind);
if strcmp(levs,'all')
  levs = [0:maxLev]';
else
  if (any(levs > maxLev) | any(levs < 0))
    error(sprintf('Level numbers must be in the range [0, %d].', maxLev));
  end
  levs = levs(:);
end

if strcmp(bands,'all')
  bands = [1:nbands]';
else
  if (any(bands < 1) | any(bands > nbands))
    error(sprintf('Band numbers must be in the range [1,3].', nbands));
  end
  bands = bands(:);
end

if (spyrHt(pind) == 0)
  if (any(levs==1))
    res1 = pyrBand(pyr,pind,2);
  else
    res1 = zeros(pind(2,:));
  end
else
  res1 = reconSpyrLevs(pyr(1+prod(pind(1,:)):size(pyr,1)), ...
      pind(2:size(pind,1),:), ...
      lofilt, bfilts, edges, levs, bands);
end

res = upConv(res1, lo0filt, edges);

%% residual highpass subband
if any(levs == 0)
   upConv( subMtx(pyr, pind(1,:)), hi0filt, edges, [1 1], [1 1], size(res), res)
;
end
```



```
% RES = reconWpyr(PYR, INDICES, FILT, EDGES, LEVS, BANDS)
%
% Reconstruct image from its separable orthonormal QMF/wavelet pyramid
% representation, as created by buildWpyr.
%
% PYR is a vector containing the N pyramid subbands, ordered from fine
% to coarse.  INDICES is an Nx2 matrix containing the sizes of
% each subband.  This is compatible with the MatLab Wavelet toolbox.
%
% FILT (optional) can be a string naming a standard filter (see
% namedFilter), or a vector which will be used for (separable)
% convolution.  Default = 'qmf9'.  EDGES specifies edge-handling,
% and defaults to 'reflect1' (see corrDn).
%
% LEVS (optional) should be a vector of levels to include, or the string
% 'all' (default).  1 corresponds to the finest scale.  The lowpass band
% corresponds to wpyrHt(INDICES)+1.
%
% BANDS (optional) should be a vector of bands to include, or the string
% 'all' (default).   1=horizontal, 2=vertical, 3=diagonal.  This is only used
% for pyramids of 2D images.

% Eero Simoncelli, 6/96.

function res = reconWpyr(pyr, ind, filt, edges, levs, bands)

if (nargin < 2)
  error('First two arguments (PYR INDICES) are required');
end

%%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('filt') ~= 1)
  filt = 'qmf9';
end

if (exist('edges') ~= 1)
  edges= 'reflect1';
end

if (exist('levs') ~= 1)
  levs = 'all';
end

if (exist('bands') ~= 1)
  bands = 'all';
end

%%------------------------------------------------------------

maxLev = 1+wpyrHt(ind);
if strcmp(levs,'all')
  levs = [1:maxLev]';
else
  if (any(levs > maxLev))
    error(sprintf('Level numbers must be in the range [1, %d].', maxLev));
  end
  levs = levs(:);
end

if strcmp(bands,'all')
  bands = [1:3]';
else
  if (any(bands < 1) | any(bands > 3))
    error('Band numbers must be in the range [1,3].');
  end
  bands = bands(:);
end

if isstr(filt)
  filt = namedFilter(filt);
end

filt = filt(:);
hfilt = modulateFlip(filt);

%% For odd-length filters, stagger the sampling lattices:
if (mod(size(filt,1),2) == 0)
        stag = 2;
else
        stag = 1;
end

%% Compute size of result image: assumes critical sampling (boundaries correct)
res_sz = ind(1,:);
if (res_sz(1) == 1)
  loind = 2;
  res_sz(2) = sum(ind(:,2));
elseif (res_sz(2) == 1)
  loind = 2;
  res_sz(1) = sum(ind(:,1));
else
  loind = 4;
  res_sz = ind(1,:) + ind(2,:);  %%horizontal + vertical bands.
  hres_sz = [ind(1,1), res_sz(2)];
  lres_sz = [ind(2,1), res_sz(2)];
end


%% First, recursively collapse coarser scales:
if any(levs > 1)

  if (size(ind,1) > loind)
    nres = reconWpyr( pyr(1+sum(prod(ind(1:loind-1,:)')):size(pyr,1)), ...
        ind(loind:size(ind,1),:), filt, edges, levs-1, bands);
  else
    nres = pyrBand(pyr, ind, loind);    % lowpass subband
  end

  if (res_sz(1) == 1)
    res = upConv(nres, filt', edges, [1 2], [1 stag], res_sz);
  elseif (res_sz(2) == 1)
    res = upConv(nres, filt, edges, [2 1], [stag 1], res_sz);
  else
    ires = upConv(nres, filt', edges, [1 2], [1 stag], lres_sz);
    res = upConv(ires, filt, edges, [2 1], [stag 1], res_sz);
  end

else

  res = zeros(res_sz);

end
```

```
%% Add  in reconstructed bands from this level:
if any(levs == 1)
  if (res_sz(1) == 1)
    upConv(pyrBand(pyr,ind,1), hfilt', edges, [1 2], [1 2], res_sz, res);
  elseif (res_sz(2) == 1)
    upConv(pyrBand(pyr,ind,1), hfilt, edges, [2 1], [2 1], res_sz, res);
  else
    if any(bands == 1) % horizontal
      ires = upConv(pyrBand(pyr,ind,1),filt',edges,[1 2],[1 stag],hres_sz);
      upConv(ires,hfilt,edges,[2 1],[2 1],res_sz,res);  %destructively modify re
s
    end
    if any(bands == 2) % vertical
      ires = upConv(pyrBand(pyr,ind,2),hfilt',edges,[1 2],[1 2],lres_sz);
      upConv(ires,filt,edges,[2 1],[stag 1],res_sz,res);  %destructively modify
res
    end
    if any(bands == 3) % diagonal
      ires =  upConv(pyrBand(pyr,ind,3),hfilt',edges,[1 2],[1 2],hres_sz);
      upConv(ires,hfilt,edges,[2 1],[2 1],res_sz,res);  %destructively modify re
s
    end
  end
end
```



```
% NEWPYR = setPyrBand(PYR, INDICES, BAND, BAND_NUM)
%
% Insert an image (BAND) into a pyramid (gaussian, laplacian, QMF/wavelet,
% or steerable).  Subbands are numbered consecutively, from finest
% (highest spatial frequency) to coarsest (lowest spatial frequency).

% Eero Simoncelli, 1/03.

function pyr =  pyrBand(pyr, pind, band, bandNum)

%% Check: PIND a valid index matrix?
if ( ~(ndims(pind) == 2) | ~(size(pind,2) == 2) | ~all(pind==round(pind)) )
  pind
  error('pyrTools:badArg',...
      'PIND argument is not an Nbands X 2 matrix of integers');
end

%% Check: PIND consistent with size of PYR?
if ( length(pyr) ~= sum(prod(pind,2)) )
  error('pyrTools:badPyr',...
      'Pyramid data vector length is inconsistent with index matrix PIND');
end

%% Check: size of BAND  consistent with desired BANDNUM?
if (~all(size(band) == pind(bandNum,:)))
  size(band)
  pind(bandNum,:)
  error('pyrTools:badArg',...
      'size of BAND to be inserted is inconsistent with BAND_NUM');
end

pyr(pyrBandIndices(pind,bandNum)) = vectify(band);
```

```
% [RES] = shift(MTX, OFFSET)
%
% Circular shift 2D matrix samples by OFFSET (a [Y,X] 2-vector),
% such that  RES(POS) = MTX(POS-OFFSET).

function res = shift(mtx, offset)

dims = size(mtx);

offset = mod(-offset,dims);

res = [ mtx(offset(1)+1:dims(1), offset(2)+1:dims(2)),  ...
          mtx(offset(1)+1:dims(1), 1:offset(2));        ...
        mtx(1:offset(1), offset(2)+1:dims(2)),          ...
          mtx(1:offset(1), 1:offset(2)) ];
```



```
% RANGE = showIm (MATRIX, RANGE, ZOOM, LABEL, NSHADES )
%
% Display a MatLab MATRIX as a grayscale image in the current figure,
% inside the current axes.  If MATRIX is complex, the real and imaginary
% parts are shown side-by-side, with the same grayscale mapping.
%
% If MATRIX is a string, it should be the name of a variable bound to a
% MATRIX in the base (global) environment.  This matrix is displayed as an
% image, with the title set to the string.
%
% RANGE (optional) is a 2-vector specifying the values that map to
% black and white, respectively.  Passing a value of 'auto' (default)
% sets RANGE=[min,max] (as in MatLab's imagesc).  'auto2' sets
% RANGE=[mean-2*stdev, mean+2*stdev].  'auto3' sets
% RANGE=[p1-(p2-p1)/8, p2+(p2-p1)/8], where p1 is the 10th percentile
% value of the sorted MATRIX samples, and p2 is the 90th percentile
% value.
%
% ZOOM specifies the number of matrix samples per screen pixel.  It
% will be rounded to an integer, or 1 divided by an integer.  A value
% of 'same' or 'auto' (default) causes the zoom value to be chosen
% automatically to fit the image into the current axes.  A value of
% 'full' fills the axis region (leaving no room for labels).  See
% pixelAxes.m.
%
% If LABEL (optional, default = 1, unless zoom='full') is non-zero, the range
% of values that are mapped into the gray colormap and the dimensions
% (size) of the matrix and zoom factor are printed below the image.  If label
% is a string, it is used as a title.
%
% NSHADES (optional) specifies the number of gray shades, and defaults
% to the size of the current colormap.

% Eero Simoncelli, 6/96.

%%TODO: should use "newplot"

function range = showIm( im, range, zoom, label, nshades );

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (nargin < 1)
  error('Requires at least one input argument.');
end

MLv = version;

if isstr(im)
  if (strcmp(MLv(1),'4'))
    error('Cannot pass string arg for MATRIX in MatLab version 4.x');
  end
  label = im;
  im = evalin('base',im);
end

if (exist('range') ~= 1)
  range = 'auto1';
end

if (exist('nshades') ~= 1)
  nshades = size(colormap,1);
end
nshades = max( nshades, 2 );

if (exist('zoom') ~= 1)
  zoom = 'auto';
end

if (exist('label') ~= 1)
  if strcmp(zoom,'full')
    label = 0;                          % no labeling
  else
    label = 1;                          % just print grayrange & dims
  end
end

%------------------------------------------------------------

%% Automatic range calculation:
if (strcmp(range,'auto1') | strcmp(range,'auto'))
  if isreal(im)
    [mn,mx] = range2(im);
  else
    [mn1,mx1] = range2(real(im));
    [mn2,mx2] =  range2(imag(im));
    mn = min(mn1,mn2);
    mx = max(mx1,mx2);
  end
  if any(size(im)==1)
    pad = (mx-mn)/12;                   % MAGIC NUMBER: graph padding
    range = [mn-pad, mx+pad];
  else
    range = [mn,mx];
  end

elseif strcmp(range,'auto2')
  if isreal(im)
    stdev = sqrt(var2(im));
    av = mean2(im);
  else
    stdev = sqrt((var2(real(im)) + var2(imag(im)))/2);
    av = (mean2(real(im)) + mean2(imag(im)))/2;
  end
  range = [av-2*stdev,av+2*stdev];      % MAGIC NUMBER: 2 stdevs

elseif strcmp(range, 'auto3')
  percentile = 0.1;                     % MAGIC NUMBER: 0<p<0.5
  [N,X] = histo(im);
  binsz = X(2)-X(1);
  N = N+1e-10;  % Ensure cumsum will be monotonic for call to interp1
  cumN = [0, cumsum(N)]/sum(N);
  cumX = [X(1)-binsz, X] + (binsz/2);
  ctrRange = interp1(cumN,cumX, [percentile, 1-percentile]);
  range = mean(ctrRange) + (ctrRange-mean(ctrRange))/(1-2*percentile);

elseif isstr(range)
  error(sprintf('Bad RANGE argument: %s',range))

end

if ((range(2) - range(1)) <= eps)
  range(1) = range(1) - 0.5;
  range(2) = range(2) + 0.5;
```

```
end


if isreal(im)
  factor=1;
else
  factor = 1+sqrt(-1);
end

xlbl_offset = 0; % default value

if (~any(size(im)==1))
  %% MatLab's "image" rounds when mapping to the colormap, so we compute
  %%      (im-r1)*(nshades-1)/(r2-r1) + 1.5
  mult = ((nshades-1) / (range(2)-range(1)));
  d_im = (mult * im) + factor*(1.5 - range(1)*mult);
end

if isreal(im)
  if (any(size(im)==1))
    hh = plot( im);
    axis([1, prod(size(im)), range]);
  else
    hh = image( d_im );
    axis('off');
    zoom = pixelAxes(size(d_im),zoom);
  end
else
  if (any(size(im)==1))
    subplot(2,1,1);
    hh = plot(real(im));
    axis([1, prod(size(im)), range]);
    subplot(2,1,2);
    hh = plot(imag(im));
    axis([1, prod(size(im)), range]);
  else
    subplot(1,2,1);
    hh = image(real(d_im));
    axis('off'); zoom = pixelAxes(size(d_im),zoom);
    ax = gca; orig_units = get(ax,'Units');
    set(ax,'Units','points');
    pos1 = get(ax,'Position');
    set(ax,'Units',orig_units);
    subplot(1,2,2);
    hh = image(imag(d_im));
    axis('off'); zoom = pixelAxes(size(d_im),zoom);
    ax = gca; orig_units = get(ax,'Units');
    set(ax,'Units','points');
    pos2 = get(ax,'Position');
    set(ax,'Units',orig_units);
    xlbl_offset = (pos1(1)-pos2(1))/2;
  end
end

if ~any(size(im)==1)
  colormap(gray(nshades));
end

if ((label ~= 0))
  if isstr(label)
    title(label);
    h = get(gca,'Title');
    orig_units = get(h,'Units');
    set(h,'Units','points');
    pos = get(h,'Position');
    pos(1:2) = pos(1:2) + [xlbl_offset, -3]; % MAGIC NUMBER: y pixel offset
    set(h,'Position',pos);
    set(h,'Units',orig_units);
  end

  if (~any(size(im)==1))
    if (zoom > 1)
      zformat = sprintf('* %d',round(zoom));
    else
      zformat = sprintf('/ %d',round(1/zoom));
    end
    if isreal(im)
      format=[' Range: [%.3g, %.3g] \n Dims: [%d, %d] ', zformat];
        else
      format=['Range: [%.3g, %.3g]  ----  Dims: [%d, %d]', zformat];
    end
    xlabel(sprintf(format, range(1), range(2), size(im,1), size(im,2)));
    h = get(gca,'Xlabel');
    set(h,'FontSize', 9);              % MAGIC NUMBER: font size!!!

    orig_units = get(h,'Units');
    set(h,'Units','points');
    pos = get(h,'Position');
    pos(1:2) = pos(1:2) + [xlbl_offset, 10]; % MAGIC NUMBER: y offset in points
    set(h,'Position',pos);
    set(h,'Units',orig_units);

    set(h,'Visible','on');             % axis('image') turned the  xlabel  off.
..
  end
end

return;
```



```
% RANGE = showLpyr (PYR, INDICES, RANGE, GAP, LEVEL_SCALE_FACTOR)
%
% Display a Laplacian (or Gaussian) pyramid, specified by PYR and
% INDICES (see buildLpyr), in the current figure.
%
% RANGE is a 2-vector specifying the values that map to black and
% white, respectively.  These values are scaled by
% LEVEL_SCALE_FACTOR^(lev-1) for bands at each level.  Passing a value
% of 'auto1' sets RANGE to the min and max values of MATRIX.  'auto2'
% sets RANGE to 3 standard deviations below and above 0.0.  In both of
% these cases, the lowpass band is independently scaled.  A value of
% 'indep1' sets the range of each subband independently, as in a call
% to showIm(subband,'auto1').  Similarly, 'indep2' causes each subband
% to be scaled independently as if by showIm(subband,'indep2').
% The default value for RANGE is 'auto1' for 1D images, and 'auto2' for
% 2D images.
%
% GAP (optional, default=1) specifies the gap in pixels to leave
% between subbands (2D images only).
%
% LEVEL_SCALE_FACTOR indicates the relative scaling between pyramid
% levels.  This should be set to the sum of the kernel taps of the
% lowpass filter used to construct the pyramid (default assumes
% L2-normalalized filters, using a value of 2 for 2D images, sqrt(2) for
% 1D images).

% Eero Simoncelli, 2/97.

function [range] = showLpyr(pyr, pind, range, gap, scale);

% Determine 1D or 2D pyramid:
if ((pind(1,1) == 1) | (pind(1,2) ==1))
  oned = 1;
else
  oned = 0;
end

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('range') ~= 1)
  if (oned==1)
    range = 'auto1';
  else
    range = 'auto2';
  end
end

if (exist('gap') ~= 1)
  gap = 1;
end

if (exist('scale') ~= 1)
  if (oned == 1)
    scale = sqrt(2);
  else
    scale = 2;
  end
end

%------------------------------------------------------------

nind = size(pind,1);

%% Auto range calculations:
if strcmp(range,'auto1')
  range = zeros(nind,1);
  mn = 0.0; mx = 0.0;
  for bnum = 1:(nind-1)
    band = pyrBand(pyr,pind,bnum)/(scale^(bnum-1));
    range(bnum) = scale^(bnum-1);
    [bmn,bmx] = range2(band);
    mn = min(mn, bmn);  mx = max(mx, bmx);
  end
  if (oned == 1)
    pad = (mx-mn)/12;                   % *** MAGIC NUMBER!!
    mn = mn-pad;  mx = mx+pad;
  end
  range = range * [mn mx];              % outer product
  band = pyrLow(pyr,pind);
  [mn,mx] = range2(band);
  if (oned == 1)
    pad = (mx-mn)/12;                   % *** MAGIC NUMBER!!
    mn = mn-pad;  mx = mx+pad;
  end
  range(nind,:) = [mn, mx];

elseif strcmp(range,'indep1')
  range = zeros(nind,2);
  for bnum = 1:nind
    band = pyrBand(pyr,pind,bnum);
    [mn,mx] = range2(band);
    if (oned == 1)
      pad = (mx-mn)/12;                 % *** MAGIC NUMBER!!
      mn = mn-pad;  mx = mx+pad;
    end
    range(bnum,:) =  [mn mx];
  end

elseif strcmp(range,'auto2')
  range = zeros(nind,1);
  sqsum = 0;  numpixels = 0;
  for bnum = 1:(nind-1)
    band = pyrBand(pyr,pind,bnum)/(scale^(bnum-1));
    sqsum = sqsum + sum(sum(band.^2));
    numpixels = numpixels + prod(size(band));
    range(bnum) = scale^(bnum-1);
  end
  stdev = sqrt(sqsum/(numpixels-1));
  range = range * [ -3*stdev 3*stdev ]; % outer product
  band = pyrLow(pyr,pind);
  av = mean2(band);   stdev = sqrt(var2(band));
  range(nind,:) = [av-2*stdev,av+2*stdev];

elseif strcmp(range,'indep2')
  range = zeros(nind,2);
  for bnum = 1:(nind-1)
    band = pyrBand(pyr,pind,bnum);
    stdev = sqrt(var2(band));
    range(bnum,:) =  [ -3*stdev 3*stdev ];
  end
  band = pyrLow(pyr,pind);
  av = mean2(band);   stdev = sqrt(var2(band));
  range(nind,:) = [av-2*stdev,av+2*stdev];
```

```
elseif isstr(range)
  error(sprintf('Bad RANGE argument: %s',range))

elseif ((size(range,1) == 1) & (size(range,2) == 2))
  scales = scale.^[0:nind-1];
  range = scales(:) * range;            % outer product
  band = pyrLow(pyr,pind);
  range(nind,:) = range(nind,:) + mean2(band) - mean(range(nind,:));

end

%% Clear Figure
clf;

if (oned == 1)

  %%%%%  1D signal:
  for bnum=1:nind
    band = pyrBand(pyr,pind,bnum);
    subplot(nind,1,nind-bnum+1);
    plot(band);
    axis([1, prod(size(band)), range(bnum,:)]);
  end

else

  %%%%% 2D signal:
  colormap(gray);
  cmap = get(gcf,'Colormap');
  nshades = size(cmap,1);

  %  Find background color index:
  clr = get(gcf,'Color');
  bg = 1;
  dist = norm(cmap(bg,:)-clr);
  for n = 1:nshades
    ndist = norm(cmap(n,:)-clr);
    if (ndist < dist)
      dist = ndist;
      bg = n;
    end
  end

  %% Compute positions of subbands:
  llpos = ones(nind,2);
  dir = [-1 -1];
  ctr = [pind(1,1)+1+gap 1];
  sz = [0 0];
  for bnum = 1:nind
    prevsz = sz;
    sz = pind(bnum,:);

    % Determine center position of new band:
    ctr = ctr + gap*dir/2 + dir.* floor((prevsz+(dir>0))/2);
    dir = dir * [0 -1; 1 0];  % ccw rotation
    ctr = ctr + gap*dir/2 + dir.* floor((sz+(dir<0))/2);
    llpos(bnum,:) = ctr - floor(sz./2);
  end

  %% Make position list positive, and allocate appropriate image:
  llpos = llpos - ones(nind,1)*min(llpos) + 1;
  urpos = llpos + pind - 1;
  d_im = bg + zeros(max(urpos));

  %% Paste bands into image, (im-r1)*(nshades-1)/(r2-r1) + 1.5
  for bnum=1:nind
    mult = (nshades-1) / (range(bnum,2)-range(bnum,1));
    d_im(llpos(bnum,1):urpos(bnum,1), llpos(bnum,2):urpos(bnum,2)) = ...
        mult*pyrBand(pyr,pind,bnum) + (1.5-mult*range(bnum,1));
  end

  hh = image(d_im);
  axis('off');
  pixelAxes(size(d_im),'full');
  set(hh,'UserData',range);

end
```



```
% RANGE = showSpyr (PYR, INDICES, RANGE, GAP, LEVEL_SCALE_FACTOR)
%
% Display a steerable pyramid, specified by PYR and INDICES
% (see buildSpyr), in the current figure.  The highpass band is not shown.
%
% RANGE is a 2-vector specifying the values that map to black and
% white, respectively.  These values are scaled by
% LEVEL_SCALE_FACTOR^(lev-1) for bands at each level.  Passing a value
% of 'auto1' sets RANGE to the min and max values of MATRIX.  'auto2'
% sets RANGE to 3 standard deviations below and above 0.0.  In both of
% these cases, the lowpass band is independently scaled.  A value of
% 'indep1' sets the range of each subband independently, as in a call
% to showIm(subband,'auto1').  Similarly, 'indep2' causes each subband
% to be scaled independently as if by showIm(subband,'indep2').
% The default value for RANGE is 'auto2'.
%
% GAP (optional, default=1) specifies the gap in pixels to leave
% between subbands.
%
% LEVEL_SCALE_FACTOR indicates the relative scaling between pyramid
% levels.  This should be set to the sum of the kernel taps of the
% lowpass filter used to construct the pyramid (default is 2, which is
% correct for L2-normalized filters.

% Eero Simoncelli, 2/97.

function [range] = showSpyr(pyr, pind, range, gap, scale);

nbands = spyrNumBands(pind);

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('range') ~= 1)
  range = 'auto2';
end

if (exist('gap') ~= 1)
  gap = 1;
end

if (exist('scale') ~= 1)
  scale = 2;
end

%------------------------------------------------------------

ht = spyrHt(pind);
nind = size(pind,1);

%% Auto range calculations:
if strcmp(range,'auto1')
  range = ones(nind,1);
  band = spyrHigh(pyr,pind);
  [mn,mx] = range2(band);
  for lnum = 1:ht
    for bnum = 1:nbands
      band = spyrBand(pyr,pind,lnum,bnum)/(scale^(lnum-1));
      range((lnum-1)*nbands+bnum+1) = scale^(lnum-1);
      [bmn,bmx] = range2(band);
      mn = min(mn, bmn);
      mx = max(mx, bmx);
    end
  end
  range = range * [mn mx];              % outer product
  band = pyrLow(pyr,pind);
  [mn,mx] = range2(band);
  range(nind,:) = [mn, mx];

elseif strcmp(range,'indep1')
  range = zeros(nind,2);
  for bnum = 1:nind
    band = pyrBand(pyr,pind,bnum);
    [mn,mx] = range2(band);
    range(bnum,:) =  [mn mx];
  end

elseif strcmp(range,'auto2')
  range = ones(nind,1);
  band = spyrHigh(pyr,pind);
  sqsum = sum(sum(band.^2));  numpixels = prod(size(band));
  for lnum = 1:ht
    for bnum = 1:nbands
      band = spyrBand(pyr,pind,lnum,bnum)/(scale^(lnum-1));
      sqsum = sqsum + sum(sum(band.^2));
      numpixels = numpixels + prod(size(band));
      range((lnum-1)*nbands+bnum+1) = scale^(lnum-1);
    end
  end
  stdev = sqrt(sqsum/(numpixels-1));
  range = range * [ -3*stdev 3*stdev ]; % outer product
  band = pyrLow(pyr,pind);
  av = mean2(band);   stdev = sqrt(var2(band));
  range(nind,:) = [av-2*stdev,av+2*stdev];

elseif strcmp(range,'indep2')
  range = zeros(nind,2);
  for bnum = 1:(nind-1)
    band = pyrBand(pyr,pind,bnum);
    stdev = sqrt(var2(band));
    range(bnum,:) =  [ -3*stdev 3*stdev ];
  end
  band = pyrLow(pyr,pind);
  av = mean2(band);   stdev = sqrt(var2(band));
  range(nind,:) = [av-2*stdev,av+2*stdev];

elseif isstr(range)
  error(sprintf('Bad RANGE argument: %s',range))

elseif ((size(range,1) == 1) & (size(range,2) == 2))
  scales = scale.^[0:(ht-1)];
  scales = ones(nbands,1) * scales;   %outer product
  scales = [1; scales(:); scale^ht];  %tack on highpass and lowpass
  range = scales * range;               % outer product
  band = pyrLow(pyr,pind);
  range(nind,:) = range(nind,:) + mean2(band) - mean(range(nind,:));

end

% CLEAR FIGURE:
clf;

colormap(gray);
cmap = get(gcf,'Colormap');
```

```
nshades = size(cmap,1);

%  Find background color index:
clr = get(gcf,'Color');
bg = 1;
dist = norm(cmap(bg,:)-clr);
for n = 1:nshades
  ndist = norm(cmap(n,:)-clr);
  if (ndist < dist)
    dist = ndist;
    bg = n;
  end
end

%% Compute positions of subbands:
llpos = ones(nind,2);

if (nbands == 2)
  ncols = 1;  nrows = 2;
else
  ncols = ceil((nbands+1)/2);   nrows = ceil(nbands/2);
end
relpos = [ (1-nrows):0, zeros(1,(ncols-1)); ...
           zeros(1,nrows), -1:-1:(1-ncols) ]';
if (nbands > 1)
  mvpos = [-1 -1];
else
  mvpos = [0 -1];
end
basepos = [0 0];

for lnum = 1:ht
  ind1 = (lnum-1)*nbands + 2;
  sz = pind(ind1,:)+gap;
  basepos = basepos + mvpos .* sz;
  if (nbands < 5)                       % to align edges...
    sz = sz + gap*(ht-lnum+1);
  end
  llpos(ind1:ind1+nbands-1,:) = relpos * diag(sz) + ones(nbands,1)*basepos;
end

% lowpass band
sz = pind(nind-1,:)+gap;
basepos = basepos + mvpos .* sz;
llpos(nind,:) = basepos;

%% Make position list positive, and allocate appropriate image:
llpos = llpos - ones(nind,1)*min(llpos) + 1;
llpos(1,:) = [1 1];
urpos = llpos + pind - 1;
d_im = bg + zeros(max(urpos));

%% Paste bands into image, (im-r1)*(nshades-1)/(r2-r1) + 1.5
for bnum=2:nind
  mult = (nshades-1) / (range(bnum,2)-range(bnum,1));
  d_im(llpos(bnum,1):urpos(bnum,1), llpos(bnum,2):urpos(bnum,2)) = ...
      mult*pyrBand(pyr,pind,bnum) + (1.5-mult*range(bnum,1));
end

hh = image(d_im);
axis('off');
pixelAxes(size(d_im),'full');
set(hh,'UserData',range);
```



```
% RANGE = showWpyr (PYR, INDICES, RANGE, GAP, LEVEL_SCALE_FACTOR)
%
% Display a separable QMF/wavelet pyramid, specified by PYR and INDICES
% (see buildWpyr), in the current figure.
%
% RANGE is a 2-vector specifying the values that map to black and
% white, respectively.  These values are scaled by
% LEVEL_SCALE_FACTOR^(lev-1) for bands at each level.  Passing a value
% of 'auto1' sets RANGE to the min and max values of MATRIX.  'auto2'
% sets RANGE to 3 standard deviations below and above 0.0.  In both of
% these cases, the lowpass band is independently scaled.  A value of
% 'indep1' sets the range of each subband independently, as in a call
% to showIm(subband,'auto1').  Similarly, 'indep2' causes each subband
% to be scaled independently as if by showIm(subband,'indep2').
% The default value for RANGE is 'auto1' for 1D images, and 'auto2' for
% 2D images.
%
% GAP (optional, default=1) specifies the gap in pixels to leave
% between subbands (2D images only).
%
% LEVEL_SCALE_FACTOR indicates the relative scaling between pyramid
% levels.  This should be set to the sum of the kernel taps of the
% lowpass filter used to construct the pyramid (default assumes
% L2-normalized filters, using a value of 2 for 2D images, sqrt(2) for
% 1D images).

% Eero Simoncelli, 2/97.

function [range] = showWpyr(pyr, pind, range, gap, scale);

% Determine 1D or 2D pyramid:
if ((pind(1,1) == 1) | (pind(1,2) ==1))
  nbands = 1;
else
  nbands = 3;
end

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('range') ~= 1)
  if (nbands==1)
    range = 'auto1';
  else
    range = 'auto2';
  end
end

if (exist('gap') ~= 1)
  gap = 1;
end

if (exist('scale') ~= 1)
  if (nbands == 1)
    scale = sqrt(2);
  else
    scale = 2;
  end
end

%------------------------------------------------------------

ht = wpyrHt(pind);
nind = size(pind,1);

%% Auto range calculations:
if strcmp(range,'auto1')
  range = zeros(nind,1);
  mn = 0.0; mx = 0.0;
  for lnum = 1:ht
    for bnum = 1:nbands
      band = wpyrBand(pyr,pind,lnum,bnum)/(scale^(lnum-1));
      range((lnum-1)*nbands+bnum) = scale^(lnum-1);
      [bmn,bmx] = range2(band);
      mn = min(mn, bmn);  mx = max(mx, bmx);
    end
  end
  if (nbands == 1)
    pad = (mx-mn)/12;                   % *** MAGIC NUMBER!!
    mn = mn-pad;  mx = mx+pad;
  end
  range = range * [mn mx];              % outer product
  band = pyrLow(pyr,pind);
  [mn,mx] = range2(band);
  if (nbands == 1)
    pad = (mx-mn)/12;                   % *** MAGIC NUMBER!!
    mn = mn-pad;  mx = mx+pad;
  end
  range(nind,:) = [mn, mx];

elseif strcmp(range,'indep1')
  range = zeros(nind,2);
  for bnum = 1:nind
    band = pyrBand(pyr,pind,bnum);
    [mn,mx] = range2(band);
    if (nbands == 1)
      pad = (mx-mn)/12;                 % *** MAGIC NUMBER!!
      mn = mn-pad;  mx = mx+pad;
    end
    range(bnum,:) =  [mn mx];
  end

elseif strcmp(range,'auto2')
  range = zeros(nind,1);
  sqsum = 0;  numpixels = 0;
  for lnum = 1:ht
    for bnum = 1:nbands
      band = wpyrBand(pyr,pind,lnum,bnum)/(scale^(lnum-1));
      sqsum = sqsum + sum(sum(band.^2));
      numpixels = numpixels + prod(size(band));
      range((lnum-1)*nbands+bnum) = scale^(lnum-1);
    end
  end
  stdev = sqrt(sqsum/(numpixels-1));
  range = range * [ -3*stdev 3*stdev ]; % outer product
  band = pyrLow(pyr,pind);
  av = mean2(band);   stdev = sqrt(var2(band));
  range(nind,:) = [av-2*stdev,av+2*stdev];

elseif strcmp(range,'indep2')
  range = zeros(nind,2);
  for bnum = 1:(nind-1)
    band = pyrBand(pyr,pind,bnum);
    stdev = sqrt(var2(band));
```

```
    range(bnum,:) =  [ -3*stdev 3*stdev ];
  end
  band = pyrLow(pyr,pind);
  av = mean2(band);   stdev = sqrt(var2(band));
  range(nind,:) = [av-2*stdev,av+2*stdev];

elseif isstr(range)
  error(sprintf('Bad RANGE argument: %s',range))

elseif ((size(range,1) == 1) & (size(range,2) == 2))
  scales = scale.^[0:ht];
  if (nbands ~= 1)
    scales = [scales; scales; scales];
  end
  range = scales(:) * range;            % outer product
  band = pyrLow(pyr,pind);
  range(nind,:) = range(nind,:) + mean2(band) - mean(range(nind,:));

end

% CLEAR FIGURE:
clf;

if (nbands == 1)

  %%%%%  1D signal:
  for bnum=1:nind
    band = pyrBand(pyr,pind,bnum);
    subplot(nind,1,nind-bnum+1);
    plot(band);
    axis([1, prod(size(band)), range(bnum,:)]);
  end

else

  %%%%% 2D signal:
  colormap(gray);
  cmap = get(gcf,'Colormap');
  nshades = size(cmap,1);

  %  Find background color index:
  clr = get(gcf,'Color');
  bg = 1;
  dist = norm(cmap(bg,:)-clr);
  for n = 1:nshades
    ndist = norm(cmap(n,:)-clr);
    if (ndist < dist)
      dist = ndist;
      bg = n;
    end
  end

  %% Compute positions of subbands:
  llpos = ones(nind,2);
  for lnum = 1:ht
    ind1 = nbands*(lnum-1) + 1;
    xpos = pind(ind1,2) + 1 + gap*(ht-lnum+1);
    ypos = pind(ind1+1,1) + 1 + gap*(ht-lnum+1);
    llpos(ind1:ind1+2,:) = [ypos 1; 1 xpos; ypos xpos];
  end
  llpos(nind,:) = [1 1];  %lowpass

  %% Make position list positive, and allocate appropriate image:
  llpos = llpos - ones(nind,1)*min(llpos) + 1;
  urpos = llpos + pind - 1;
  d_im = bg + zeros(max(urpos));

  %% Paste bands into image, (im-r1)*(nshades-1)/(r2-r1) + 1.5
  for bnum=1:nind
    mult = (nshades-1) / (range(bnum,2)-range(bnum,1));
    d_im(llpos(bnum,1):urpos(bnum,1), llpos(bnum,2):urpos(bnum,2)) = ...
        mult*pyrBand(pyr,pind,bnum) + (1.5-mult*range(bnum,1));
  end

  hh = image(d_im);
  axis('off');
  pixelAxes(size(d_im),'full');
  set(hh,'UserData',range);

end
```



```
% S = SKEW2(MTX,MEAN,VAR)
%
% Sample skew (third moment divided by variance^3/2) of a matrix.
%  MEAN (optional) and VAR (optional) make the computation faster.

function res = skew2(mtx, mn, v)

if (exist('mn') ~= 1)
  mn =  mean2(mtx);
end

if (exist('v') ~= 1)
  v =  var2(mtx,mn);
end

if (isreal(mtx))
  res = mean(mean((mtx-mn).^3)) / (v^(3/2));
else
  res = mean(mean(real(mtx-mn).^3)) / (real(v)^(3/2)) + ...
      i * mean(mean(imag(mtx-mn).^3)) / (imag(v)^(3/2));
end
```

```
% Steerable pyramid filters.  Transform described  in:
%
% @INPROCEEDINGS{Simoncelli95b,
%       TITLE = "The Steerable Pyramid: A Flexible Architecture for
%                Multi-Scale Derivative Computation",
%       AUTHOR = "E P Simoncelli and W T Freeman",
%       BOOKTITLE = "Second Int'l Conf on Image Processing",
%       ADDRESS = "Washington, DC", MONTH = "October", YEAR = 1995 }
%
% Filter kernel design described in:
%
%@INPROCEEDINGS{Karasaridis96,
%       TITLE = "A Filter Design Technique for
%                Steerable Pyramid Image Transforms",
%       AUTHOR = "A Karasaridis and E P Simoncelli",
%       BOOKTITLE = "ICASSP",    ADDRESS = "Atlanta, GA",
%       MONTH = "May",  YEAR = 1996 }

% Eero Simoncelli, 6/96.

function [lo0filt,hi0filt,lofilt,bfilts,mtx,harmonics] = sp0Filters();

harmonics = [ 0 ];

lo0filt = [ ...
-4.514000e-04 -1.137100e-04 -3.725800e-04 -3.743860e-03 -3.725800e-04 -1.137100e
-04 -4.514000e-04
-1.137100e-04 -6.119520e-03 -1.344160e-02 -7.563200e-03 -1.344160e-02 -6.119520e
-03 -1.137100e-04
-3.725800e-04 -1.344160e-02 6.441488e-02 1.524935e-01 6.441488e-02 -1.344160e-02
 -3.725800e-04
-3.743860e-03 -7.563200e-03 1.524935e-01 3.153017e-01 1.524935e-01 -7.563200e-03
 -3.743860e-03
-3.725800e-04 -1.344160e-02 6.441488e-02 1.524935e-01 6.441488e-02 -1.344160e-02
 -3.725800e-04
-1.137100e-04 -6.119520e-03 -1.344160e-02 -7.563200e-03 -1.344160e-02 -6.119520e
-03 -1.137100e-04
-4.514000e-04 -1.137100e-04 -3.725800e-04 -3.743860e-03 -3.725800e-04 -1.137100e
-04 -4.514000e-04];

lofilt = [ ...
-2.257000e-04 -8.064400e-04 -5.686000e-05 8.741400e-04 -1.862800e-04 -1.031640e-
03 -1.871920e-03 -1.031640e-03 -1.862800e-04 8.741400e-04 -5.686000e-05 -8.06440
0e-04 -2.257000e-04
-8.064400e-04 1.417620e-03 -1.903800e-04 -2.449060e-03 -4.596420e-03 -7.006740e-
03 -6.948900e-03 -7.006740e-03 -4.596420e-03 -2.449060e-03 -1.903800e-04 1.41762
0e-03 -8.064400e-04
-5.686000e-05 -1.903800e-04 -3.059760e-03 -6.401000e-03 -6.720800e-03 -5.236180e
-03 -3.781600e-03 -5.236180e-03 -6.720800e-03 -6.401000e-03 -3.059760e-03 -1.903
800e-04 -5.686000e-05
8.741400e-04 -2.449060e-03 -6.401000e-03 -5.260020e-03 3.938620e-03 1.722078e-02
 2.449600e-02 1.722078e-02 3.938620e-03 -5.260020e-03 -6.401000e-03 -2.449060e-0
3 8.741400e-04
-1.862800e-04 -4.596420e-03 -6.720800e-03 3.938620e-03 3.220744e-02 6.306262e-02
 7.624674e-02 6.306262e-02 3.220744e-02 3.938620e-03 -6.720800e-03 -4.596420e-03
 -1.862800e-04
-1.031640e-03 -7.006740e-03 -5.236180e-03 1.722078e-02 6.306262e-02 1.116388e-01
 1.348999e-01 1.116388e-01 6.306262e-02 1.722078e-02 -5.236180e-03 -7.006740e-03
 -1.031640e-03
-1.871920e-03 -6.948900e-03 -3.781600e-03 2.449600e-02 7.624674e-02 1.348999e-01
 1.576508e-01 1.348999e-01 7.624674e-02 2.449600e-02 -3.781600e-03 -6.948900e-03
 -1.871920e-03
-1.031640e-03 -7.006740e-03 -5.236180e-03 1.722078e-02 6.306262e-02 1.116388e-01
 1.348999e-01 1.116388e-01 6.306262e-02 1.722078e-02 -5.236180e-03 -7.006740e-03
 -1.031640e-03
-1.862800e-04 -4.596420e-03 -6.720800e-03 3.938620e-03 3.220744e-02 6.306262e-02
 7.624674e-02 6.306262e-02 3.220744e-02 3.938620e-03 -6.720800e-03 -4.596420e-03
 -1.862800e-04
8.741400e-04 -2.449060e-03 -6.401000e-03 -5.260020e-03 3.938620e-03 1.722078e-02
 2.449600e-02 1.722078e-02 3.938620e-03 -5.260020e-03 -6.401000e-03 -2.449060e-0
3 8.741400e-04
-5.686000e-05 -1.903800e-04 -3.059760e-03 -6.401000e-03 -6.720800e-03 -5.236180e
-03 -3.781600e-03 -5.236180e-03 -6.720800e-03 -6.401000e-03 -3.059760e-03 -1.903
800e-04 -5.686000e-05
-8.064400e-04 1.417620e-03 -1.903800e-04 -2.449060e-03 -4.596420e-03 -7.006740e-
03 -6.948900e-03 -7.006740e-03 -4.596420e-03 -2.449060e-03 -1.903800e-04 1.41762
0e-03 -8.064400e-04
-2.257000e-04 -8.064400e-04 -5.686000e-05 8.741400e-04 -1.862800e-04 -1.031640e-
03 -1.871920e-03 -1.031640e-03 -1.862800e-04 8.741400e-04 -5.686000e-05 -8.06440
0e-04 -2.257000e-04];

mtx = [ 1.000000 ];

hi0filt = [...
5.997200e-04 -6.068000e-05 -3.324900e-04 -3.325600e-04 -2.406600e-04 -3.325600e-
04 -3.324900e-04 -6.068000e-05 5.997200e-04
-6.068000e-05 1.263100e-04 4.927100e-04 1.459700e-04 -3.732100e-04 1.459700e-04
4.927100e-04 1.263100e-04 -6.068000e-05
-3.324900e-04 4.927100e-04 -1.616650e-03 -1.437358e-02 -2.420138e-02 -1.437358e-
02 -1.616650e-03 4.927100e-04 -3.324900e-04
-3.325600e-04 1.459700e-04 -1.437358e-02 -6.300923e-02 -9.623594e-02 -6.300923e-
02 -1.437358e-02 1.459700e-04 -3.325600e-04
-2.406600e-04 -3.732100e-04 -2.420138e-02 -9.623594e-02 8.554893e-01 -9.623594e-
02 -2.420138e-02 -3.732100e-04 -2.406600e-04
-3.325600e-04 1.459700e-04 -1.437358e-02 -6.300923e-02 -9.623594e-02 -6.300923e-
02 -1.437358e-02 1.459700e-04 -3.325600e-04
-3.324900e-04 4.927100e-04 -1.616650e-03 -1.437358e-02 -2.420138e-02 -1.437358e-
02 -1.616650e-03 4.927100e-04 -3.324900e-04
-6.068000e-05 1.263100e-04 4.927100e-04 1.459700e-04 -3.732100e-04 1.459700e-04
4.927100e-04 1.263100e-04 -6.068000e-05
5.997200e-04 -6.068000e-05 -3.324900e-04 -3.325600e-04 -2.406600e-04 -3.325600e-
04 -3.324900e-04 -6.068000e-05 5.997200e-04 ];

bfilts = [ ...
-9.066000e-05 -1.738640e-03 -4.942500e-03 -7.889390e-03 -1.009473e-02 -7.889390e
-03 -4.942500e-03 -1.738640e-03 -9.066000e-05 ...
-1.738640e-03 -4.625150e-03 -7.272540e-03 -7.623410e-03 -9.091950e-03 -7.623410e
-03 -7.272540e-03 -4.625150e-03 -1.738640e-03 ...
-4.942500e-03 -7.272540e-03 -2.129540e-02 -2.435662e-02 -3.487008e-02 -2.435662e
-02 -2.129540e-02 -7.272540e-03 -4.942500e-03  ...
-7.889390e-03 -7.623410e-03 -2.435662e-02 -1.730466e-02 -3.158605e-02 -1.730466e
-02 -2.435662e-02 -7.623410e-03 -7.889390e-03  ...
-1.009473e-02 -9.091950e-03 -3.487008e-02 -3.158605e-02 9.464195e-01 -3.158605e-
02 -3.487008e-02 -9.091950e-03 -1.009473e-02  ...
-7.889390e-03 -7.623410e-03 -2.435662e-02 -1.730466e-02 -3.158605e-02 -1.730466e
-02 -2.435662e-02 -7.623410e-03 -7.889390e-03 ...
-4.942500e-03 -7.272540e-03 -2.129540e-02 -2.435662e-02 -3.487008e-02 -2.435662e
-02 -2.129540e-02 -7.272540e-03 -4.942500e-03  ...
-1.738640e-03 -4.625150e-03 -7.272540e-03 -7.623410e-03 -9.091950e-03 -7.623410e
-03 -7.272540e-03 -4.625150e-03 -1.738640e-03  ...
-9.066000e-05 -1.738640e-03 -4.942500e-03 -7.889390e-03 -1.009473e-02 -7.889390e
-03 -4.942500e-03 -1.738640e-03 -9.066000e-05 ]';
```



```
% Steerable pyramid filters.  Transform described  in:
%
% @INPROCEEDINGS{Simoncelli95b,
%       TITLE = "The Steerable Pyramid: A Flexible Architecture for
%                Multi-Scale Derivative Computation",
%       AUTHOR = "E P Simoncelli and W T Freeman",
%       BOOKTITLE = "Second Int'l Conf on Image Processing",
%       ADDRESS = "Washington, DC", MONTH = "October", YEAR = 1995 }
%
% Filter kernel design described in:
%
%@INPROCEEDINGS{Karasaridis96,
%       TITLE = "A Filter Design Technique for
%                Steerable Pyramid Image Transforms",
%       AUTHOR = "A Karasaridis and E P Simoncelli",
%       BOOKTITLE = "ICASSP",    ADDRESS = "Atlanta, GA",
%       MONTH = "May",  YEAR = 1996 }

% Eero Simoncelli, 6/96.

function [lo0filt,hi0filt,lofilt,bfilts,mtx,harmonics] = sp1Filters();

harmonics = [ 1 ];

%% filters only contain first harmonic.
mtx = eye(2);

lo0filt = [ ...
-8.701000e-05 -1.354280e-03 -1.601260e-03 -5.033700e-04 2.524010e-03 -5.033700e-
04 -1.601260e-03 -1.354280e-03 -8.701000e-05
-1.354280e-03 2.921580e-03 7.522720e-03 8.224420e-03 1.107620e-03 8.224420e-03 7
.522720e-03 2.921580e-03 -1.354280e-03
-1.601260e-03 7.522720e-03 -7.061290e-03 -3.769487e-02 -3.297137e-02 -3.769487e-
02 -7.061290e-03 7.522720e-03 -1.601260e-03
-5.033700e-04 8.224420e-03 -3.769487e-02 4.381320e-02 1.811603e-01 4.381320e-02
-3.769487e-02 8.224420e-03 -5.033700e-04
2.524010e-03 1.107620e-03 -3.297137e-02 1.811603e-01 4.376250e-01 1.811603e-01 -
3.297137e-02 1.107620e-03 2.524010e-03
-5.033700e-04 8.224420e-03 -3.769487e-02 4.381320e-02 1.811603e-01 4.381320e-02
-3.769487e-02 8.224420e-03 -5.033700e-04
-1.601260e-03 7.522720e-03 -7.061290e-03 -3.769487e-02 -3.297137e-02 -3.769487e-
02 -7.061290e-03 7.522720e-03 -1.601260e-03
-1.354280e-03 2.921580e-03 7.522720e-03 8.224420e-03 1.107620e-03 8.224420e-03 7
.522720e-03 2.921580e-03 -1.354280e-03
-8.701000e-05 -1.354280e-03 -1.601260e-03 -5.033700e-04 2.524010e-03 -5.033700e-
04 -1.601260e-03 -1.354280e-03 -8.701000e-05
];

lofilt = [ ...
-4.350000e-05 1.207800e-04 -6.771400e-04 -1.243400e-04 -8.006400e-04 -1.597040e-
03 -2.516800e-04 -4.202000e-04 1.262000e-03 -4.202000e-04 -2.516800e-04 -1.59704
0e-03 -8.006400e-04 -1.243400e-04 -6.771400e-04 1.207800e-04 -4.350000e-05  ; ..
.
1.207800e-04 4.460600e-04 -5.814600e-04 5.621600e-04 -1.368800e-04 2.325540e-03
2.889860e-03 4.287280e-03 5.589400e-03 4.287280e-03 2.889860e-03 2.325540e-03 -1
.368800e-04 5.621600e-04 -5.814600e-04 4.460600e-04 1.207800e-04  ; ...
-6.771400e-04 -5.814600e-04 1.460780e-03 2.160540e-03 3.761360e-03 3.080980e-03
4.112200e-03 2.221220e-03 5.538200e-04 2.221220e-03 4.112200e-03 3.080980e-03 3.
761360e-03 2.160540e-03 1.460780e-03 -5.814600e-04 -6.771400e-04  ; ...
-1.243400e-04 5.621600e-04 2.160540e-03 3.175780e-03 3.184680e-03 -1.777480e-03
-7.431700e-03 -9.056920e-03 -9.637220e-03 -9.056920e-03 -7.431700e-03 -1.777480e
-03 3.184680e-03 3.175780e-03 2.160540e-03 5.621600e-04 -1.243400e-04  ; ...
-8.006400e-04 -1.368800e-04 3.761360e-03 3.184680e-03 -3.530640e-03 -1.260420e-0
2 -1.884744e-02 -1.750818e-02 -1.648568e-02 -1.750818e-02 -1.884744e-02 -1.26042
0e-02 -3.530640e-03 3.184680e-03 3.761360e-03 -1.368800e-04 -8.006400e-04  ; ...
-1.597040e-03 2.325540e-03 3.080980e-03 -1.777480e-03 -1.260420e-02 -2.022938e-0
2 -1.109170e-02 3.955660e-03 1.438512e-02 3.955660e-03 -1.109170e-02 -2.022938e-
02 -1.260420e-02 -1.777480e-03 3.080980e-03 2.325540e-03 -1.597040e-03  ; ...
-2.516800e-04 2.889860e-03 4.112200e-03 -7.431700e-03 -1.884744e-02 -1.109170e-0
2 2.190660e-02 6.806584e-02 9.058014e-02 6.806584e-02 2.190660e-02 -1.109170e-02
 -1.884744e-02 -7.431700e-03 4.112200e-03 2.889860e-03 -2.516800e-04  ; ...
-4.202000e-04 4.287280e-03 2.221220e-03 -9.056920e-03 -1.750818e-02 3.955660e-03
 6.806584e-02 1.445500e-01 1.773651e-01 1.445500e-01 6.806584e-02 3.955660e-03 -
1.750818e-02 -9.056920e-03 2.221220e-03 4.287280e-03 -4.202000e-04  ; ...
1.262000e-03 5.589400e-03 5.538200e-04 -9.637220e-03 -1.648568e-02 1.438512e-02
9.058014e-02 1.773651e-01 2.120374e-01 1.773651e-01 9.058014e-02 1.438512e-02 -1
.648568e-02 -9.637220e-03 5.538200e-04 5.589400e-03 1.262000e-03  ; ...
-4.202000e-04 4.287280e-03 2.221220e-03 -9.056920e-03 -1.750818e-02 3.955660e-03
 6.806584e-02 1.445500e-01 1.773651e-01 1.445500e-01 6.806584e-02 3.955660e-03 -
1.750818e-02 -9.056920e-03 2.221220e-03 4.287280e-03 -4.202000e-04  ; ...
-2.516800e-04 2.889860e-03 4.112200e-03 -7.431700e-03 -1.884744e-02 -1.109170e-0
2 2.190660e-02 6.806584e-02 9.058014e-02 6.806584e-02 2.190660e-02 -1.109170e-02
 -1.884744e-02 -7.431700e-03 4.112200e-03 2.889860e-03 -2.516800e-04  ; ...
-1.597040e-03 2.325540e-03 3.080980e-03 -1.777480e-03 -1.260420e-02 -2.022938e-0
2 -1.109170e-02 3.955660e-03 1.438512e-02 3.955660e-03 -1.109170e-02 -2.022938e-
02 -1.260420e-02 -1.777480e-03 3.080980e-03 2.325540e-03 -1.597040e-03  ; ...
-8.006400e-04 -1.368800e-04 3.761360e-03 3.184680e-03 -3.530640e-03 -1.260420e-0
2 -1.884744e-02 -1.750818e-02 -1.648568e-02 -1.750818e-02 -1.884744e-02 -1.26042
0e-02 -3.530640e-03 3.184680e-03 3.761360e-03 -1.368800e-04 -8.006400e-04  ; ...
-1.243400e-04 5.621600e-04 2.160540e-03 3.175780e-03 3.184680e-03 -1.777480e-03
-7.431700e-03 -9.056920e-03 -9.637220e-03 -9.056920e-03 -7.431700e-03 -1.777480e
-03 3.184680e-03 3.175780e-03 2.160540e-03 5.621600e-04 -1.243400e-04  ; ...
-6.771400e-04 -5.814600e-04 1.460780e-03 2.160540e-03 3.761360e-03 3.080980e-03
4.112200e-03 2.221220e-03 5.538200e-04 2.221220e-03 4.112200e-03 3.080980e-03 3.
761360e-03 2.160540e-03 1.460780e-03 -5.814600e-04 -6.771400e-04  ; ...
1.207800e-04 4.460600e-04 -5.814600e-04 5.621600e-04 -1.368800e-04 2.325540e-03
2.889860e-03 4.287280e-03 5.589400e-03 4.287280e-03 2.889860e-03 2.325540e-03 -1
.368800e-04 5.621600e-04 -5.814600e-04 4.460600e-04 1.207800e-04  ; ...
-4.350000e-05 1.207800e-04 -6.771400e-04 -1.243400e-04 -8.006400e-04 -1.597040e-
03 -2.516800e-04 -4.202000e-04 1.262000e-03 -4.202000e-04 -2.516800e-04 -1.59704
0e-03 -8.006400e-04 -1.243400e-04 -6.771400e-04 1.207800e-04 -4.350000e-05 ];

hi0filt = [...
-9.570000e-04 -2.424100e-04 -1.424720e-03 -8.742600e-04 -1.166810e-03 -8.742600e
-04 -1.424720e-03 -2.424100e-04 -9.570000e-04 ; ...
-2.424100e-04 -4.317530e-03 8.998600e-04 9.156420e-03 1.098012e-02 9.156420e-03
8.998600e-04 -4.317530e-03 -2.424100e-04  ; ...
-1.424720e-03 8.998600e-04 1.706347e-02 1.094866e-02 -5.897780e-03 1.094866e-02
1.706347e-02 8.998600e-04 -1.424720e-03  ; ...
-8.742600e-04 9.156420e-03 1.094866e-02 -7.841370e-02 -1.562827e-01 -7.841370e-0
2 1.094866e-02 9.156420e-03 -8.742600e-04  ; ...
-1.166810e-03 1.098012e-02 -5.897780e-03 -1.562827e-01 7.282593e-01 -1.562827e-0
1 -5.897780e-03 1.098012e-02 -1.166810e-03  ; ...
-8.742600e-04 9.156420e-03 1.094866e-02 -7.841370e-02 -1.562827e-01 -7.841370e-0
2 1.094866e-02 9.156420e-03 -8.742600e-04  ; ...
-1.424720e-03 8.998600e-04 1.706347e-02 1.094866e-02 -5.897780e-03 1.094866e-02
1.706347e-02 8.998600e-04 -1.424720e-03  ; ...
-2.424100e-04 -4.317530e-03 8.998600e-04 9.156420e-03 1.098012e-02 9.156420e-03
8.998600e-04 -4.317530e-03 -2.424100e-04  ; ...
-9.570000e-04 -2.424100e-04 -1.424720e-03 -8.742600e-04 -1.166810e-03 -8.742600e
-04 -1.424720e-03 -2.424100e-04 -9.570000e-04 ];

bfilts = -[ ...
6.125880e-03 -8.052600e-03 -2.103714e-02 -1.536890e-02 -1.851466e-02 -1.536890e-
```

```
02 −2.103714e−02 −8.052600e−03 6.125880e−03  ...
−1.287416e−02 −9.611520e−03 1.023569e−02 6.009450e−03 1.872620e−03 6.009450e−03
1.023569e−02 −9.611520e−03 −1.287416e−02  ...
−5.641530e−03 4.168400e−03 −2.382180e−02 −5.375324e−02 −2.076086e−02 −5.375324e−
02 −2.382180e−02 4.168400e−03 −5.641530e−03  ...
−8.957260e−03 −1.751170e−03 −1.836909e−02 1.265655e−01 2.996168e−01 1.265655e−01
 −1.836909e−02 −1.751170e−03 −8.957260e−03  ...
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.
000000e+00 0.000000e+00 0.000000e+00  ...
8.957260e−03 1.751170e−03 1.836909e−02 −1.265655e−01 −2.996168e−01 −1.265655e−01
 1.836909e−02 1.751170e−03 8.957260e−03  ...
5.641530e−03 −4.168400e−03 2.382180e−02 5.375324e−02 2.076086e−02 5.375324e−02 2
.382180e−02 −4.168400e−03 5.641530e−03  ...
1.287416e−02 9.611520e−03 −1.023569e−02 −6.009450e−03 −1.872620e−03 −6.009450e−0
3 −1.023569e−02 9.611520e−03 1.287416e−02  ...
−6.125880e−03 8.052600e−03 2.103714e−02 1.536890e−02 1.851466e−02 1.536890e−02 2
.103714e−02 8.052600e−03 −6.125880e−03; ...
...
−6.125880e−03 1.287416e−02 5.641530e−03 8.957260e−03 0.000000e+00 −8.957260e−03
−5.641530e−03 −1.287416e−02 6.125880e−03  ...
8.052600e−03 9.611520e−03 −4.168400e−03 1.751170e−03 0.000000e+00 −1.751170e−03
4.168400e−03 −9.611520e−03 −8.052600e−03  ...
2.103714e−02 −1.023569e−02 2.382180e−02 1.836909e−02 0.000000e+00 −1.836909e−02
−2.382180e−02 1.023569e−02 −2.103714e−02  ...
1.536890e−02 −6.009450e−03 5.375324e−02 −1.265655e−01 0.000000e+00 1.265655e−01
−5.375324e−02 6.009450e−03 −1.536890e−02  ...
1.851466e−02 −1.872620e−03 2.076086e−02 −2.996168e−01 0.000000e+00 2.996168e−01
−2.076086e−02 1.872620e−03 −1.851466e−02  ...
1.536890e−02 −6.009450e−03 5.375324e−02 −1.265655e−01 0.000000e+00 1.265655e−01
−5.375324e−02 6.009450e−03 −1.536890e−02  ...
2.103714e−02 −1.023569e−02 2.382180e−02 1.836909e−02 0.000000e+00 −1.836909e−02
−2.382180e−02 1.023569e−02 −2.103714e−02  ...
8.052600e−03 9.611520e−03 −4.168400e−03 1.751170e−03 0.000000e+00 −1.751170e−03
4.168400e−03 −9.611520e−03 −8.052600e−03  ...
−6.125880e−03 1.287416e−02 5.641530e−03 8.957260e−03 0.000000e+00 −8.957260e−03
−5.641530e−03 −1.287416e−02 6.125880e−03 ...
]’;
```



```
% Steerable pyramid filters.  Transform described  in:
%
% @INPROCEEDINGS{Simoncelli95b,
%       TITLE = "The Steerable Pyramid: A Flexible Architecture for
%                Multi−Scale Derivative Computation",
%       AUTHOR = "E P Simoncelli and W T Freeman",
%       BOOKTITLE = "Second Int’l Conf on Image Processing",
%       ADDRESS = "Washington, DC", MONTH = "October", YEAR = 1995 }
%
% Filter kernel design described in:
%
%@INPROCEEDINGS{Karasaridis96,
%       TITLE = "A Filter Design Technique for
%                Steerable Pyramid Image Transforms",
%       AUTHOR = "A Karasaridis and E P Simoncelli",
%       BOOKTITLE = "ICASSP",   ADDRESS = "Atlanta, GA",
%       MONTH = "May",  YEAR = 1996 }

% Eero Simoncelli, 6/96.

function [lo0filt,hi0filt,lofilt,bfilts,mtx,harmonics] = sp3Filters();

harmonics = [1 3];

mtx = [ ...
    0.5000    0.3536         0   −0.3536
   −0.0000    0.3536    0.5000    0.3536
    0.5000   −0.3536         0    0.3536
   −0.0000    0.3536   −0.5000    0.3536];

hi0filt = [
−4.0483998600E−4  −6.2596000498E−4  −3.7829999201E−5  8.8387000142E−4  1.5450799
838E−3  1.9235999789E−3  2.0687500946E−3  2.0898699295E−3  2.0687500946E−3  1.92
35999789E−3  1.5450799838E−3  8.8387000142E−4  −3.7829999201E−5  −6.2596000498E−
4  −4.0483998600E−4
−6.2596000498E−4  −3.2734998967E−4  7.7435001731E−4  1.5874400269E−3  2.17507011
26E−3  2.5626500137E−3  2.2892199922E−3  1.9755100366E−3  2.2892199922E−3  2.562
6500137E−3  2.1750701126E−3  1.5874400269E−3  7.7435001731E−4  −3.2734998967E−4
 −6.2596000498E−4
−3.7829999201E−5  7.7435001731E−4  1.1793200392E−3  1.4050999889E−3  2.225340111
2E−3  2.1145299543E−3  3.3578000148E−4  −8.3368999185E−4  3.3578000148E−4  2.114
5299543E−3  2.2253401112E−3  1.4050999889E−3  1.1793200392E−3  7.7435001731E−4
−3.7829999201E−5
8.8387000142E−4  1.5874400269E−3  1.4050999889E−3  1.2960999738E−3  −4.927400150
3E−4  −3.1295299996E−3  −4.5751798898E−3  −5.1014497876E−3  −4.5751798898E−3  −3
.1295299996E−3  −4.9274001503E−4  1.2960999738E−3  1.4050999889E−3  1.5874400269
E−3  8.8387000142E−4
1.5450799838E−3  2.1750701126E−3  2.2253401112E−3  −4.9274001503E−4  −6.32226979
36E−3  −2.7556000277E−3  5.3632198833E−3  7.3032598011E−3  5.3632198833E−3  −2.7
556000277E−3  −6.3222697936E−3  −4.9274001503E−4  2.2253401112E−3  2.1750701126E
−3  1.5450799838E−3
1.9235999789E−3  2.5626500137E−3  2.1145299543E−3  −3.1295299996E−3  −2.75560002
77E−3  1.3962360099E−2  7.8046298586E−3  −9.3812197447E−3  7.8046298586E−3  1.39
62360099E−2  −2.7556000277E−3  −3.1295299996E−3  2.1145299543E−3  2.5626500137E−
3  1.9235999789E−3
2.0687500946E−3  2.2892199922E−3  3.3578000148E−4  −4.5751798898E−3  5.363219883
3E−3  7.8046298586E−3  −7.9501636326E−2  −0.1554141641      −7.9501636326E−2  7.
8046298586E−3  5.3632198833E−3  −4.5751798898E−3  3.3578000148E−4  2.2892199922E
−3  2.0687500946E−3
2.0898699295E−3  1.9755100366E−3  −8.3368999185E−4  −5.1014497876E−3  7.30325980
11E−3  −9.3812197447E−3  −0.1554141641      0.7303866148      −0.1554141641
 −9.3812197447E−3  7.3032598011E−3  −5.1014497876E−3  −8.3368999185E−4  1.975510
0366E−3  2.0898699295E−3
2.0687500946E−3  2.2892199922E−3  3.3578000148E−4  −4.5751798898E−3  5.363219883
3E−3  7.8046298586E−3  −7.9501636326E−2  −0.1554141641      −7.9501636326E−2  7.
8046298586E−3  5.3632198833E−3  −4.5751798898E−3  3.3578000148E−4  2.2892199922E
−3  2.0687500946E−3
1.9235999789E−3  2.5626500137E−3  2.1145299543E−3  −3.1295299996E−3  −2.75560002
77E−3  1.3962360099E−2  7.8046298586E−3  −9.3812197447E−3  7.8046298586E−3  1.39
62360099E−2  −2.7556000277E−3  −3.1295299996E−3  2.1145299543E−3  2.5626500137E−
3  1.9235999789E−3
1.5450799838E−3  2.1750701126E−3  2.2253401112E−3  −4.9274001503E−4  −6.32226979
36E−3  −2.7556000277E−3  5.3632198833E−3  7.3032598011E−3  5.3632198833E−3  −2.7
556000277E−3  −6.3222697936E−3  −4.9274001503E−4  2.2253401112E−3  2.1750701126E
−3  1.5450799838E−3
8.8387000142E−4  1.5874400269E−3  1.4050999889E−3  1.2960999738E−3  −4.927400150
3E−4  −3.1295299996E−3  −4.5751798898E−3  −5.1014497876E−3  −4.5751798898E−3  −3
.1295299996E−3  −4.9274001503E−4  1.2960999738E−3  1.4050999889E−3  1.5874400269
E−3  8.8387000142E−4
−3.7829999201E−5  7.7435001731E−4  1.1793200392E−3  1.4050999889E−3  2.225340111
2E−3  2.1145299543E−3  3.3578000148E−4  −8.3368999185E−4  3.3578000148E−4  2.114
5299543E−3  2.2253401112E−3  1.4050999889E−3  1.1793200392E−3  7.7435001731E−4
−3.7829999201E−5
−6.2596000498E−4  −3.2734998967E−4  7.7435001731E−4  1.5874400269E−3  2.17507011
26E−3  2.5626500137E−3  2.2892199922E−3  1.9755100366E−3  2.2892199922E−3  2.562
6500137E−3  2.1750701126E−3  1.5874400269E−3  7.7435001731E−4  −3.2734998967E−4
 −6.2596000498E−4
−4.0483998600E−4  −6.2596000498E−4  −3.7829999201E−5  8.8387000142E−4  1.5450799
838E−3  1.9235999789E−3  2.0687500946E−3  2.0898699295E−3  2.0687500946E−3  1.92
35999789E−3  1.5450799838E−3  8.8387000142E−4  −3.7829999201E−5  −6.2596000498E−
4  −4.0483998600E−4
];

lo0filt = [
−8.7009997515E−5  −1.3542800443E−3  −1.6012600390E−3  −5.0337001448E−4  2.524009
9058E−3  −5.0337001448E−4  −1.6012600390E−3  −1.3542800443E−3  −8.7009997515E−5
−1.3542800443E−3  2.9215801042E−3  7.5227199122E−3  8.2244202495E−3  1.107619958
9E−3  8.2244202495E−3  7.5227199122E−3  2.9215801042E−3  −1.3542800443E−3
−1.6012600390E−3  7.5227199122E−3  −7.0612900890E−3  −3.7694871426E−2  −3.297137
0965E−2  −3.7694871426E−2  −7.0612900890E−3  7.5227199122E−3  −1.6012600390E−3
−5.0337001448E−4  8.2244202495E−3  −3.7694871426E−2  4.3813198805E−2  0.18116030
10      4.3813198805E−2  −3.7694871426E−2  8.2244202495E−3  −5.0337001448E−4
2.5240099058E−3  1.1076199589E−3  −3.2971370965E−2  0.1811603010      0.43762499
09      0.1811603010      −3.2971370965E−2  1.1076199589E−3  2.5240099058E−3
−5.0337001448E−4  8.2244202495E−3  −3.7694871426E−2  4.3813198805E−2  0.18116030
10      4.3813198805E−2  −3.7694871426E−2  8.2244202495E−3  −5.0337001448E−4
−1.6012600390E−3  7.5227199122E−3  −7.0612900890E−3  −3.7694871426E−2  −3.297137
0965E−2  −3.7694871426E−2  −7.0612900890E−3  7.5227199122E−3  −1.6012600390E−3
−1.3542800443E−3  2.9215801042E−3  7.5227199122E−3  8.2244202495E−3  1.107619958
9E−3  8.2244202495E−3  7.5227199122E−3  2.9215801042E−3  −1.3542800443E−3
−8.7009997515E−5  −1.3542800443E−3  −1.6012600390E−3  −5.0337001448E−4  2.524009
9058E−3  −5.0337001448E−4  −1.6012600390E−3  −1.3542800443E−3  −8.7009997515E−5
];

lofilt = [
−4.3500000174E−5  1.2078000145E−4  −6.7714002216E−4  −1.2434000382E−4  −8.006399
7302E−4  −1.5970399836E−3  −2.5168000138E−4  −4.2019999819E−4  1.2619999470E−3
−4.2019999819E−4  −2.5168000138E−4  −1.5970399836E−3  −8.0063997302E−4  −1.24340
00382E−4  −6.7714002216E−4  1.2078000145E−4  −4.3500000174E−5
1.2078000145E−4  4.4606000301E−4  −5.8146001538E−4  5.6215998484E−4  −1.36880000
35E−4  2.3255399428E−3  2.8898599558E−3  4.2872801423E−3  5.5893999524E−3  4.287
2801423E−3  2.8898599558E−3  2.3255399428E−3  −1.3688000035E−4  5.6215998484E−4
 −5.8146001538E−4  4.4606000301E−4  1.2078000145E−4
−6.7714002216E−4  −5.8146001538E−4  1.4607800404E−3  2.1605400834E−3  3.76135995
```

```
61E-3  3.0809799209E-3  4.1121998802E-3  2.2212199401E-3  5.5381999118E-4  2.221
2199401E-3  4.1121998802E-3  3.0809799209E-3  3.7613599561E-3  2.1605400834E-3
1.4607800404E-3  -5.8146001538E-4  -6.7714002216E-4
-1.2434000382E-4  5.6215998484E-4  2.1605400834E-3  3.1757799443E-3  3.184679895
6E-3  -1.7774800071E-3  -7.4316998944E-3  -9.0569201857E-3  -9.6372198313E-3  -9
.0569201857E-3  -7.4316998944E-3  -1.7774800071E-3  3.1846798956E-3  3.175779944
3E-3  2.1605400834E-3  5.6215998484E-4  -1.2434000382E-4
-8.0063997302E-4  -1.3688000035E-4  3.7613599561E-3  3.1846798956E-3  -3.5306399
222E-3  -1.2604200281E-2  -1.8847439438E-2  -1.7508180812E-2  -1.6485679895E-2
-1.7508180812E-2  -1.8847439438E-2  -1.2604200281E-2  -3.5306399222E-3  3.184679
8956E-3  3.7613599561E-3  -1.3688000035E-4  -8.0063997302E-4
-1.5970399836E-3  2.3255399428E-3  3.0809799209E-3  -1.7774800071E-3  -1.2604200
281E-2  -2.0229380578E-2  -1.1091699824E-2  3.9556599222E-3  1.4385120012E-2  3.
9556599222E-3  -1.1091699824E-2  -2.0229380578E-2  -1.2604200281E-2  -1.77748000
71E-3  3.0809799209E-3  2.3255399428E-3  -1.5970399836E-3
-2.5168000138E-4  2.8898599558E-3  4.1121998802E-3  -7.4316998944E-3  -1.8847439
438E-2  -1.1091699824E-2  2.1906599402E-2  6.8065837026E-2  9.0580143034E-2  6.8
065837026E-2  2.1906599402E-2  -1.1091699824E-2  -1.8847439438E-2  -7.4316998944
E-3  4.1121998802E-3  2.8898599558E-3  -2.5168000138E-4
-4.2019999819E-4  4.2872801423E-3  2.2212199401E-3  -9.0569201857E-3  -1.7508180
812E-2  3.9556599222E-3  6.8065837026E-2  0.1445499808      0.1773651242      0.
1445499808      6.8065837026E-2  3.9556599222E-3  -1.7508180812E-2  -9.056920185
7E-3  2.2212199401E-3  4.2872801423E-3  -4.2019999819E-4
1.2619999470E-3  5.5893999524E-3  5.5381999118E-4  -9.6372198313E-3  -1.64856798
95E-2  1.4385120012E-2  9.0580143034E-2  0.1773651242      0.2120374441      0.1
773651242      9.0580143034E-2  1.4385120012E-2  -1.6485679895E-2  -9.6372198313
E-3  5.5381999118E-4  5.5893999524E-3  1.2619999470E-3
-4.2019999819E-4  4.2872801423E-3  2.2212199401E-3  -9.0569201857E-3  -1.7508180
812E-2  3.9556599222E-3  6.8065837026E-2  0.1445499808      0.1773651242      0.
1445499808      6.8065837026E-2  3.9556599222E-3  -1.7508180812E-2  -9.056920185
7E-3  2.2212199401E-3  4.2872801423E-3  -4.2019999819E-4
-2.5168000138E-4  2.8898599558E-3  4.1121998802E-3  -7.4316998944E-3  -1.8847439
438E-2  -1.1091699824E-2  2.1906599402E-2  6.8065837026E-2  9.0580143034E-2  6.8
065837026E-2  2.1906599402E-2  -1.1091699824E-2  -1.8847439438E-2  -7.4316998944
E-3  4.1121998802E-3  2.8898599558E-3  -2.5168000138E-4
-1.5970399836E-3  2.3255399428E-3  3.0809799209E-3  -1.7774800071E-3  -1.2604200
281E-2  -2.0229380578E-2  -1.1091699824E-2  3.9556599222E-3  1.4385120012E-2  3.
9556599222E-3  -1.1091699824E-2  -2.0229380578E-2  -1.2604200281E-2  -1.77748000
71E-3  3.0809799209E-3  2.3255399428E-3  -1.5970399836E-3
-8.0063997302E-4  -1.3688000035E-4  3.7613599561E-3  3.1846798956E-3  -3.5306399
222E-3  -1.2604200281E-2  -1.8847439438E-2  -1.7508180812E-2  -1.6485679895E-2
-1.7508180812E-2  -1.8847439438E-2  -1.2604200281E-2  -3.5306399222E-3  3.184679
8956E-3  3.7613599561E-3  -1.3688000035E-4  -8.0063997302E-4
-1.2434000382E-4  5.6215998484E-4  2.1605400834E-3  3.1757799443E-3  3.184679895
6E-3  -1.7774800071E-3  -7.4316998944E-3  -9.0569201857E-3  -9.6372198313E-3  -9
.0569201857E-3  -7.4316998944E-3  -1.7774800071E-3  3.1846798956E-3  3.175779944
3E-3  2.1605400834E-3  5.6215998484E-4  -1.2434000382E-4
-6.7714002216E-4  -5.8146001538E-4  1.4607800404E-3  2.1605400834E-3  3.76135995
61E-3  3.0809799209E-3  4.1121998802E-3  2.2212199401E-3  5.5381999118E-4  2.221
2199401E-3  4.1121998802E-3  3.0809799209E-3  3.7613599561E-3  2.1605400834E-3
1.4607800404E-3  -5.8146001538E-4  -6.7714002216E-4
1.2078000145E-4  4.4606000301E-4  -5.8146001538E-4  5.6215998484E-4  -1.36880000
35E-4  2.3255399428E-3  2.8898599558E-3  4.2872801423E-3  5.5893999524E-3  4.287
2801423E-3  2.8898599558E-3  2.3255399428E-3  -1.3688000035E-4  5.6215998484E-4
 -5.8146001538E-4  4.4606000301E-4  1.2078000145E-4
-4.3500000174E-5  1.2078000145E-4  -6.7714002216E-4  -1.2434000382E-4  -8.006399
7302E-4  -1.5970399836E-3  -2.5168000138E-4  -4.2019999819E-4  1.2619999470E-3
-4.2019999819E-4  -2.5168000138E-4  -1.5970399836E-3  -8.0063997302E-4  -1.24340
00382E-4  -6.7714002216E-4  1.2078000145E-4  -4.3500000174E-5
];

bfilts = [...
-8.1125000725E-4  4.4451598078E-3  1.2316980399E-2  1.3955879956E-2  1.417945046
0E-2  1.3955879956E-2  1.2316980399E-2  4.4451598078E-3  -8.1125000725E-4 ...
3.9103501476E-3  4.4565401040E-3  -5.8724298142E-3  -2.8760801069E-3  8.52676015
35E-3  -2.8760801069E-3  -5.8724298142E-3  4.4565401040E-3  3.9103501476E-3 ...
1.3462699717E-3  -3.7740699481E-3  8.2581602037E-3  3.9442278445E-2  5.360563844
4E-2  3.9442278445E-2  8.2581602037E-3  -3.7740699481E-3  1.3462699717E-3 ...
7.4700999539E-4  -3.6522001028E-4  -2.2522680461E-2  -0.1105690673      -0.17684
19296      -0.1105690673      -2.2522680461E-2  -3.6522001028E-4  7.4700999539E-
4 ...
0.0000000000      0.0000000000      0.0000000000      0.0000000000      0.000000
0000      0.0000000000      0.0000000000      0.0000000000      0.0000000000
 ...
-7.4700999539E-4  3.6522001028E-4  2.2522680461E-2  0.1105690673      0.17684192
96      0.1105690673      2.2522680461E-2  3.6522001028E-4  -7.4700999539E-4 ...
-1.3462699717E-3  3.7740699481E-3  -8.2581602037E-3  -3.9442278445E-2  -5.360563
8444E-2  -3.9442278445E-2  -8.2581602037E-3  3.7740699481E-3  -1.3462699717E-3 .
..
-3.9103501476E-3  -4.4565401040E-3  5.8724298142E-3  2.8760801069E-3  -8.5267601
535E-3  2.8760801069E-3  5.8724298142E-3  -4.4565401040E-3  -3.9103501476E-3 ...
8.1125000725E-4  -4.4451598078E-3  -1.2316980399E-2  -1.3955879956E-2  -1.417945
0460E-2  -1.3955879956E-2  -1.2316980399E-2  -4.4451598078E-3  8.1125000725E-4;
...
...
0.0000000000      -8.2846998703E-4  -5.7109999034E-5  4.0110000555E-5  4.6670897
864E-3  8.0871898681E-3  1.4807609841E-2  8.6204400286E-3  -3.1221499667E-3 ...
8.2846998703E-4  0.0000000000      -9.7479997203E-4  -6.9718998857E-3  -2.086560
0090E-3  2.3298799060E-3  -4.4814897701E-3  1.4917500317E-2  8.6204400286E-3 ...
5.7109999034E-5  9.7479997203E-4  0.0000000000      -1.2145539746E-2  -2.4427289
143E-2  5.0797060132E-2  3.2785870135E-2  -4.4814897701E-3  1.4807609841E-2 ...
-4.0110000555E-5  6.9718998857E-3  1.2145539746E-2  0.0000000000      -0.1510555
595      -8.2495503128E-2  5.0797060132E-2  2.3298799060E-3  8.0871898681E-3 ...
-4.6670897864E-3  2.0865600090E-3  2.4427289143E-2  0.1510555595      0.00000000
00      -0.1510555595      -2.4427289143E-2  -2.0865600090E-3  4.6670897864E-3 .
..
-8.0871898681E-3  -2.3298799060E-3  -5.0797060132E-2  8.2495503128E-2  0.1510555
595      0.0000000000      -1.2145539746E-2  -6.9718998857E-3  4.0110000555E-5 .
..
-1.4807609841E-2  4.4814897701E-3  -3.2785870135E-2  -5.0797060132E-2  2.4427289
143E-2  1.2145539746E-2  0.0000000000      -9.7479997203E-4  -5.7109999034E-5 ..
.
-8.6204400286E-3  -1.4917500317E-2  4.4814897701E-3  -2.3298799060E-3  2.0865600
090E-3  6.9718998857E-3  9.7479997203E-4  0.0000000000      -8.2846998703E-4 ...
3.1221499667E-3  -8.6204400286E-3  -1.4807609841E-2  -8.0871898681E-3  -4.667089
7864E-3  -4.0110000555E-5  5.7109999034E-5  8.2846998703E-4  0.0000000000; ...
...
8.1125000725E-4  -3.9103501476E-3  -1.3462699717E-3  -7.4700999539E-4  0.0000000
000      7.4700999539E-4  1.3462699717E-3  3.9103501476E-3  -8.1125000725E-4 ...
-4.4451598078E-3  -4.4565401040E-3  3.7740699481E-3  3.6522001028E-4  0.00000000
00      -3.6522001028E-4  -3.7740699481E-3  4.4565401040E-3  4.4451598078E-3 ...
-1.2316980399E-2  5.8724298142E-3  -8.2581602037E-3  2.2522680461E-2  0.00000000
00      -2.2522680461E-2  8.2581602037E-3  -5.8724298142E-3  1.2316980399E-2 ...
-1.3955879956E-2  2.8760801069E-3  -3.9442278445E-2  0.1105690673      0.0000000
000      -0.1105690673      3.9442278445E-2  -2.8760801069E-3  1.3955879956E-2 .
..
-1.4179450460E-2  -8.5267601535E-3  -5.3605638444E-2  0.1768419296      0.000000
0000      -0.1768419296      5.3605638444E-2  8.5267601535E-3  1.4179450460E-2 ..
.
-1.3955879956E-2  2.8760801069E-3  -3.9442278445E-2  0.1105690673      0.0000000
000      -0.1105690673      3.9442278445E-2  -2.8760801069E-3  1.3955879956E-2 .
..
-1.2316980399E-2  5.8724298142E-3  -8.2581602037E-3  2.2522680461E-2  0.00000000
00      -2.2522680461E-2  8.2581602037E-3  -5.8724298142E-3  1.2316980399E-2 ...
```



```
-4.4451598078E-3  -4.4565401040E-3  3.7740699481E-3  3.6522001028E-4  0.00000000
00      -3.6522001028E-4  -3.7740699481E-3  4.4565401040E-3  4.4451598078E-3 ...
8.1125000725E-4  -3.9103501476E-3  -1.3462699717E-3  -7.4700999539E-4  0.0000000
000      7.4700999539E-4  1.3462699717E-3  3.9103501476E-3  -8.1125000725E-4; ..
.
...
3.1221499667E-3  -8.6204400286E-3  -1.4807609841E-2  -8.0871898681E-3  -4.667089
7864E-3  -4.0110000555E-5  5.7109999034E-5  8.2846998703E-4  0.0000000000  ...

-8.6204400286E-3  -1.4917500317E-2  4.4814897701E-3  -2.3298799060E-3  2.0865600
090E-3  6.9718998857E-3  9.7479997203E-4  -0.0000000000      -8.2846998703E-4 ..
.
-1.4807609841E-2  4.4814897701E-3  -3.2785870135E-2  -5.0797060132E-2  2.4427289
143E-2  1.2145539746E-2  0.0000000000      -9.7479997203E-4  -5.7109999034E-5 ..
.
-8.0871898681E-3  -2.3298799060E-3  -5.0797060132E-2  8.2495503128E-2  0.1510555
595      -0.0000000000      -1.2145539746E-2  -6.9718998857E-3  4.0110000555E-5
...
-4.6670897864E-3  2.0865600090E-3  2.4427289143E-2  0.1510555595      0.00000000
00      -0.1510555595      -2.4427289143E-2  -2.0865600090E-3  4.6670897864E-3 .
..
-4.0110000555E-5  6.9718998857E-3  1.2145539746E-2  0.0000000000      -0.1510555
595      -8.2495503128E-2  5.0797060132E-2  2.3298799060E-3  8.0871898681E-3 ...
5.7109999034E-5  9.7479997203E-4  -0.0000000000      -1.2145539746E-2  -2.442728
9143E-2  5.0797060132E-2  3.2785870135E-2  -4.4814897701E-3  1.4807609841E-2 ...
8.2846998703E-4  -0.0000000000      -9.7479997203E-4  -6.9718998857E-3  -2.08656
00090E-3  2.3298799060E-3  -4.4814897701E-3  1.4917500317E-2  8.6204400286E-3 ..
.
0.0000000000      -8.2846998703E-4  -5.7109999034E-5  4.0110000555E-5  4.6670897
864E-3  8.0871898681E-3  1.4807609841E-2  8.6204400286E-3  -3.1221499667E-3 ...
]';
```

```
% Steerable pyramid filters.  Transform described  in:
%
% @INPROCEEDINGS{Simoncelli95b,
%       TITLE = "The Steerable Pyramid: A Flexible Architecture for
%                Multi-Scale Derivative Computation",
%       AUTHOR = "E P Simoncelli and W T Freeman",
%       BOOKTITLE = "Second Int'l Conf on Image Processing",
%       ADDRESS = "Washington, DC", MONTH = "October", YEAR = 1995 }
%
% Filter kernel design described in:
%
%@INPROCEEDINGS{Karasaridis96,
%       TITLE = "A Filter Design Technique for
%                Steerable Pyramid Image Transforms",
%       AUTHOR = "A Karasaridis and E P Simoncelli",
%       BOOKTITLE = "ICASSP",    ADDRESS = "Atlanta, GA",
%       MONTH = "May",  YEAR = 1996 }

% Eero Simoncelli, 6/96.

function [lo0filt,hi0filt,lofilt,bfilts,mtx,harmonics] = sp5Filters();

harmonics = [1 3 5];

mtx = [ ...
    0.3333    0.2887    0.1667    0.0000   -0.1667   -0.2887
    0.0000    0.1667    0.2887    0.3333    0.2887    0.1667
    0.3333   -0.0000   -0.3333   -0.0000    0.3333   -0.0000
    0.0000    0.3333    0.0000   -0.3333    0.0000    0.3333
    0.3333   -0.2887    0.1667   -0.0000   -0.1667    0.2887
   -0.0000    0.1667   -0.2887    0.3333   -0.2887    0.1667];

hi0filt = [
-0.00033429 -0.00113093 -0.00171484 -0.00133542 -0.00080639 -0.00133542 -0.00171
484 -0.00113093 -0.00033429
-0.00113093 -0.00350017 -0.00243812 0.00631653 0.01261227 0.00631653 -0.00243812
 -0.00350017 -0.00113093
-0.00171484 -0.00243812 -0.00290081 -0.00673482 -0.00981051 -0.00673482 -0.00290
081 -0.00243812 -0.00171484
-0.00133542 0.00631653 -0.00673482 -0.07027679 -0.11435863 -0.07027679 -0.006734
82 0.00631653 -0.00133542
-0.00080639 0.01261227 -0.00981051 -0.11435863 0.81380200 -0.11435863 -0.0098105
1 0.01261227 -0.00080639
-0.00133542 0.00631653 -0.00673482 -0.07027679 -0.11435863 -0.07027679 -0.006734
82 0.00631653 -0.00133542
-0.00171484 -0.00243812 -0.00290081 -0.00673482 -0.00981051 -0.00673482 -0.00290
081 -0.00243812 -0.00171484
-0.00113093 -0.00350017 -0.00243812 0.00631653 0.01261227 0.00631653 -0.00243812
 -0.00350017 -0.00113093
-0.00033429 -0.00113093 -0.00171484 -0.00133542 -0.00080639 -0.00133542 -0.00171
484 -0.00113093 -0.00033429];


lo0filt = [
0.00341614 -0.01551246 -0.03848215 -0.01551246 0.00341614
-0.01551246 0.05586982 0.15925570 0.05586982 -0.01551246
-0.03848215 0.15925570 0.40304148 0.15925570 -0.03848215
-0.01551246 0.05586982 0.15925570 0.05586982 -0.01551246
0.00341614 -0.01551246 -0.03848215 -0.01551246 0.00341614];

lofilt = 2*[
0.00085404 -0.00244917 -0.00387812 -0.00944432 -0.00962054 -0.00944432 -0.003878
12 -0.00244917 0.00085404
-0.00244917 -0.00523281 -0.00661117 0.00410600 0.01002988 0.00410600 -0.00661117
 -0.00523281 -0.00244917
-0.00387812 -0.00661117 0.01396746 0.03277038 0.03981393 0.03277038 0.01396746 -
0.00661117 -0.00387812
-0.00944432 0.00410600 0.03277038 0.06426333 0.08169618 0.06426333 0.03277038 0.
00410600 -0.00944432
-0.00962054 0.01002988 0.03981393 0.08169618 0.10096540 0.08169618 0.03981393 0.
01002988 -0.00962054
-0.00944432 0.00410600 0.03277038 0.06426333 0.08169618 0.06426333 0.03277038 0.
00410600 -0.00944432
-0.00387812 -0.00661117 0.01396746 0.03277038 0.03981393 0.03277038 0.01396746 -
0.00661117 -0.00387812
-0.00244917 -0.00523281 -0.00661117 0.00410600 0.01002988 0.00410600 -0.00661117
 -0.00523281 -0.00244917
0.00085404 -0.00244917 -0.00387812 -0.00944432 -0.00962054 -0.00944432 -0.003878
12 -0.00244917 0.00085404];

bfilts = [...
 0.00277643 0.00496194 0.01026699 0.01455399 0.01026699 0.00496194 0.00277643 ..
.
-0.00986904 -0.00893064 0.01189859 0.02755155 0.01189859 -0.00893064 -0.00986904
 ...
-0.01021852 -0.03075356 -0.08226445 -0.11732297 -0.08226445 -0.03075356 -0.01021
852 ...
 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 ..
.
 0.01021852 0.03075356 0.08226445 0.11732297 0.08226445 0.03075356 0.01021852 ..
.
 0.00986904 0.00893064 -0.01189859 -0.02755155 -0.01189859 0.00893064 0.00986904
 ...
-0.00277643 -0.00496194 -0.01026699 -0.01455399 -0.01026699 -0.00496194 -0.00277
643;
...
-0.00343249 -0.00640815 -0.00073141 0.01124321 0.00182078 0.00285723 0.01166982
...
-0.00358461 -0.01977507 -0.04084211 -0.00228219 0.03930573 0.01161195 0.00128000
 ...
 0.01047717 0.01486305 -0.04819057 -0.12227230 -0.05394139 0.00853965 -0.0045903
4 ...
 0.00790407 0.04435647 0.09454202 -0.00000000 -0.09454202 -0.04435647 -0.0079040
7 ...
 0.00459034 -0.00853965 0.05394139 0.12227230 0.04819057 -0.01486305 -0.01047717
 ...
-0.00128000 -0.01161195 -0.03930573 0.00228219 0.04084211 0.01977507 0.00358461
...
-0.01166982 -0.00285723 -0.00182078 -0.01124321 0.00073141 0.00640815 0.00343249
;
...
 0.00343249 0.00358461 -0.01047717 -0.00790407 -0.00459034 0.00128000 0.01166982
 ...
 0.00640815 0.01977507 -0.01486305 -0.04435647 0.00853965 0.01161195 0.00285723
...
 0.00073141 0.04084211 0.04819057 -0.09454202 -0.05394139 0.03930573 0.00182078
...
-0.01124321 0.00228219 0.12227230 -0.00000000 -0.12227230 -0.00228219 0.01124321
 ...
-0.00182078 -0.03930573 0.05394139 0.09454202 -0.04819057 -0.04084211 -0.0007314
1 ...
-0.00285723 -0.01161195 -0.00853965 0.04435647 0.01486305 -0.01977507 -0.0064081
5 ...
-0.01166982 -0.00128000 0.00459034 0.00790407 0.01047717 -0.00358461 -0.00343249
```



```
;
...
-0.00277643 0.00986904 0.01021852 -0.00000000 -0.01021852 -0.00986904 0.00277643
 ...
-0.00496194 0.00893064 0.03075356 -0.00000000 -0.03075356 -0.00893064 0.00496194
 ...
-0.01026699 -0.01189859 0.08226445 -0.00000000 -0.08226445 0.01189859 0.01026699
 ...
-0.01455399 -0.02755155 0.11732297 -0.00000000 -0.11732297 0.02755155 0.01455399
 ...
-0.01026699 -0.01189859 0.08226445 -0.00000000 -0.08226445 0.01189859 0.01026699
 ...
-0.00496194 0.00893064 0.03075356 -0.00000000 -0.03075356 -0.00893064 0.00496194
 ...
-0.00277643 0.00986904 0.01021852 -0.00000000 -0.01021852 -0.00986904 0.00277643
;
...
-0.01166982 -0.00128000 0.00459034 0.00790407 0.01047717 -0.00358461 -0.00343249
 ...
-0.00285723 -0.01161195 -0.00853965 0.04435647 0.01486305 -0.01977507 -0.0064081
5 ...
-0.00182078 -0.03930573 0.05394139 0.09454202 -0.04819057 -0.04084211 -0.0007314
1 ...
-0.01124321 0.00228219 0.12227230 -0.00000000 -0.12227230 -0.00228219 0.01124321
 ...
 0.00073141 0.04084211 0.04819057 -0.09454202 -0.05394139 0.03930573 0.00182078
...
 0.00640815 0.01977507 -0.01486305 -0.04435647 0.00853965 0.01161195 0.00285723
...
 0.00343249 0.00358461 -0.01047717 -0.00790407 -0.00459034 0.00128000 0.01166982
;
...
-0.01166982 -0.00285723 -0.00182078 -0.01124321 0.00073141 0.00640815 0.00343249
 ...
-0.00128000 -0.01161195 -0.03930573 0.00228219 0.04084211 0.01977507 0.00358461
...
 0.00459034 -0.00853965 0.05394139 0.12227230 0.04819057 -0.01486305 -0.01047717
 ...
 0.00790407 0.04435647 0.09454202 -0.00000000 -0.09454202 -0.04435647 -0.0079040
7 ...
 0.01047717 0.01486305 -0.04819057 -0.12227230 -0.05394139 0.00853965 -0.0045903
4 ...
-0.00358461 -0.01977507 -0.04084211 -0.00228219 0.03930573 0.01161195 0.00128000
 ...
-0.00343249 -0.00640815 -0.00073141 0.01124321 0.00182078 0.00285723 0.01166982]
';
```

```
% [LEV,IND] = spyrBand(PYR,INDICES,LEVEL,BAND)
%
% Access a band from a steerable pyramid.
%
%   LEVEL indicates the scale (finest = 1, coarsest = spyrHt(INDICES)).
%
%   BAND (optional, default=1) indicates which subband
%     (1 = vertical, rest proceeding anti−clockwise).

% Eero Simoncelli, 6/96.

function res =  spyrBand(pyr,pind,level,band)

if (exist('level') ~= 1)
  level = 1;
end

if (exist('band') ~= 1)
  band = 1;
end

nbands = spyrNumBands(pind);
if ((band > nbands) | (band < 1))
  error(sprintf('Bad band number (%d) should be in range [1,%d].', band, nbands)
);
end

maxLev = spyrHt(pind);
if ((level > maxLev) | (level < 1))
  error(sprintf('Bad level number (%d), should be in range [1,%d].', level, maxL
ev));
end

firstband = 1 + band + nbands*(level−1);
res  = pyrBand(pyr, pind, firstband);
```



```
% RES = spyrHigh(PYR, INDICES)
%
% Access the highpass residual band from a steerable pyramid.

% Eero Simoncelli, 6/96.

function res =  spyrHigh(pyr,pind)

res  = pyrBand(pyr, pind, 1);
```

```
% [HEIGHT] = spyrHt(INDICES)
%
% Compute height of steerable pyramid with given index matrix.

% Eero Simoncelli, 6/96.

function [ht] =  spyrHt(pind)

nbands = spyrNumBands(pind);

% Don't count lowpass, or highpass residual bands
if (size(pind,1) > 2)
  ht = (size(pind,1)-2)/nbands;
else
  ht = 0;
end
```



```
% [LEV,IND] = spyrLev(PYR,INDICES,LEVEL)
%
% Access a level from a steerable pyramid.
% Return as an SxB matrix, B = number of bands, S = total size of a band.
% Also returns an Bx2 matrix containing dimensions of the subbands.

% Eero Simoncelli, 6/96.

function [lev,ind] =  spyrLev(pyr,pind,level)

nbands = spyrNumBands(pind);

if ((level > spyrHt(pind)) | (level < 1))
  error(sprintf('Level number must be in the range [1, %d].', spyrHt(pind)));
end

firstband = 2 + nbands*(level-1);
firstind = 1;
for l=1:firstband-1
  firstind = firstind + prod(pind(l,:));
end

ind = pind(firstband:firstband+nbands-1,:);
lev  = pyr(firstind:firstind+sum(prod(ind'))-1);
```

```
% [NBANDS] = spyrNumBands(INDICES)
%
% Compute number of orientation bands in a steerable pyramid with
% given index matrix.  If the pyramid contains only the highpass and
% lowpass bands (i.e., zero levels), returns 0.

% Eero Simoncelli, 2/97.

function [nbands] =  spyrNumBands(pind)

if (size(pind,1) == 2)
  nbands  = 0;
else
  % Count number of orientation bands:
  b = 3;
  while ((b <= size(pind,1)) & all( pind(b,:) == pind(2,:)) )
    b = b+1;
  end
  nbands = b-2;
end
```



```
% MTX = steer2HarmMtx(HARMONICS, ANGLES, REL_PHASES)
%
% Compute a steering matrix (maps a directional basis set onto the
% angular Fourier harmonics).  HARMONICS is a vector specifying the
% angular harmonics contained in the steerable basis/filters.  ANGLES
% (optional) is a vector specifying the angular position of each filter.
% REL_PHASES (optional, default = 'even') specifies whether the harmonics
% are cosine or sine phase aligned about those positions.
% The result matrix is suitable for passing to the function STEER.

% Eero Simoncelli, 7/96.

function mtx = steer2HarmMtx(harmonics, angles, evenorodd)

%%=================================================================
%%% Optional Parameters:

if (exist('evenorodd') ~= 1)
  evenorodd = 'even';
end

% Make HARMONICS a row vector
harmonics = harmonics(:)';

numh = 2*size(harmonics,2) - any(harmonics == 0);

if (exist('angles') ~= 1)
  angles = pi * [0:numh-1]'/numh;
else
  angles = angles(:);
end

%%=================================================================

if isstr(evenorodd)
  if strcmp(evenorodd,'even')
    evenorodd = 0;
  elseif strcmp(evenorodd,'odd')
    evenorodd = 1;
  else
    error('EVEN_OR_ODD should be the string  EVEN or ODD');
  end
end

%% Compute inverse matrix, which maps Fourier components onto
%% steerable basis.
imtx = zeros(size(angles,1),numh);
col = 1;
for h=harmonics
  args = h*angles;
  if (h == 0)
    imtx(:,col) = ones(size(angles));
    col = col+1;
  elseif evenorodd
    imtx(:,col) = sin(args);
    imtx(:,col+1) = -cos(args);
    col = col+2;
  else
    imtx(:,col) = cos(args);
    imtx(:,col+1) = sin(args);
    col = col+2;
  end
end

r = rank(imtx);
if (( r ~= numh ) & ( r ~= size(angles,1) ))
  fprintf(2,'WARNING: matrix is not full rank');
end

mtx = pinv(imtx);
```

```
% RES = STEER(BASIS, ANGLE, HARMONICS, STEERMTX)
%
% Steer BASIS to the specfied ANGLE.
%
% BASIS should be a matrix whose columns are vectorized rotated copies of a
% steerable function, or the responses of a set of steerable filters.
%
% ANGLE can be a scalar, or a column vector the size of the basis.
%
% HARMONICS (optional, default is N even or odd low frequencies, as for
% derivative filters) should be a list of harmonic numbers indicating
% the angular harmonic content of the basis.
%
% STEERMTX (optional, default assumes cosine phase harmonic components,
% and filter positions at 2pi*n/N) should be a matrix which maps
% the filters onto Fourier series components (ordered [cos0 cos1 sin1
% cos2 sin2 ... sinN]).  See steer2HarmMtx.m

% Eero Simoncelli, 7/96.

function res = steer(basis,angle,harmonics,steermtx)

num = size(basis,2);

if ( any(size(angle) ~= [size(basis,1) 1]) & any(size(angle) ~= [1 1]) )
  error('ANGLE must be a scalar, or a column vector the size of the basis elemen
ts');
end

%% If HARMONICS are not passed, assume derivatives.
if (exist('harmonics') ~= 1)
  if (mod(num,2) == 0)
    harmonics = [0:(num/2)-1]'*2 + 1;
  else
    harmonics = [0:(num-1)/2]'*2;
  end
else
  harmonics = harmonics(:);
  if ((2*size(harmonics,1)-any(harmonics == 0)) ~= num)
    error('harmonics list is incompatible with basis size');
  end
end

%% If STEERMTX not passed, assume evenly distributed cosine-phase filters:
if (exist('steermtx') ~= 1)
  steermtx = steer2HarmMtx(harmonics, pi*[0:num-1]/num, 'even');
end

steervect = zeros(size(angle,1),num);
arg = angle * harmonics(find(harmonics~=0))';
if (all(harmonics))
        steervect(:, 1:2:num) = cos(arg);
        steervect(:, 2:2:num) = sin(arg);
else
        steervect(:, 1) = ones(size(arg,1),1);
        steervect(:, 2:2:num) = cos(arg);
        steervect(:, 3:2:num) = sin(arg);
end

steervect = steervect * steermtx;

if (size(steervect,1) > 1)
        tmp = basis' .* steervect';
        res = sum(tmp)';
else
        res = basis * steervect';
end
```



```
% MTX = subMtx(VEC, DIMENSIONS, START_INDEX)
%
% Reshape a portion of VEC starting from START_INDEX (optional,
% default=1) to the given dimensions.

% Eero Simoncelli, 6/96.

function mtx = subMtx(vec, sz, offset)

if (exist('offset') ~= 1)
   offset = 1;
end

vec = vec(:);
sz = sz(:);

if (size(sz,1) ~= 2)
  error('DIMENSIONS must be a 2-vector.');
end

mtx = reshape( vec(offset:offset+prod(sz)-1), sz(1), sz(2) );
```

```
% RES = upBlur(IM, LEVELS, FILT)
%
% Upsample and blur an image.  The blurring is done with filter
% kernel specified by FILT (default = 'binom5'), which can be a string
% (to be passed to namedFilter), a vector (applied separably as a 1D
% convolution kernel in X and Y), or a matrix (applied as a 2D
% convolution kernel).  The downsampling is always by 2 in each
% direction.
%
% The procedure is applied recursively LEVELS times (default=1).

% Eero Simoncelli, 4/97.

function res = upBlur(im, nlevs, filt)

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('nlevs') ~= 1)
  nlevs = 1;
end

if (exist('filt') ~= 1)
  filt = 'binom5';
end

%------------------------------------------------------------

if isstr(filt)
  filt = namedFilter(filt);
end

if nlevs > 1
  im = upBlur(im,nlevs-1,filt);
end

if (nlevs >= 1)
  if (any(size(im)==1))
    if (size(im,1)==1)
      filt = filt';
    end
    res = upConv(im,filt,'reflect1',(size(im)~=1)+1);
  elseif (any(size(filt)==1))
    filt = filt(:);
    res = upConv(im,filt,'reflect1',[2 1]);
    res = upConv(res,filt','reflect1',[1 2]);
  else
    res = upConv(im,filt,'reflect1',[2 2]);
  end
else
  res = im;
end
```



```
% RES = upConv(IM, FILT, EDGES, STEP, START, STOP, RES)
%
% Upsample matrix IM, followed by convolution with matrix FILT.  These
% arguments should be 1D or 2D matrices, and IM must be larger (in
% both dimensions) than FILT.  The origin of filt
% is assumed to be floor(size(filt)/2)+1.
%
% EDGES is a string determining boundary handling:
%    'circular' - Circular convolution
%    'reflect1' - Reflect about the edge pixels
%    'reflect2' - Reflect, doubling the edge pixels
%    'repeat'   - Repeat the edge pixels
%    'zero'     - Assume values of zero outside image boundary
%    'extend'   - Reflect and invert
%    'dont-compute' - Zero output when filter overhangs OUTPUT boundaries
%
% Upsampling factors are determined by STEP (optional, default=[1 1]),
% a 2-vector [y,x].
%
% The window over which the convolution occurs is specfied by START
% (optional, default=[1,1], and STOP (optional, default =
% step .* (size(IM) + floor((start-1)./step))).
%
% RES is an optional result matrix.  The convolution result will be
% destructively added into this matrix.  If this argument is passed, the
% result matrix will not be returned. DO NOT USE THIS ARGUMENT IF
% YOU DO NOT UNDERSTAND WHAT THIS MEANS!!
%
% NOTE: this operation corresponds to multiplication of a signal
% vector by a matrix whose columns contain copies of the time-reversed
% (or space-reversed) FILT shifted by multiples of STEP.  See corrDn.m
% for the operation corresponding to the transpose of this matrix.

% Eero Simoncelli, 6/96.  revised 2/97.

function result = upConv(im,filt,edges,step,start,stop,res)

%% THIS CODE IS NOT ACTUALLY USED! (MEX FILE IS CALLED INSTEAD)

fprintf(1,'WARNING: You should compile the MEX version of "upConv.c",\n         f
ound in the MEX subdirectory of matlabPyrTools, and put it in your matlab path.
 It is MUCH faster, and provides more boundary-handling options.\n');

%------------------------------------------------------------
%% OPTIONAL ARGS:

if (exist('edges') == 1)
  if (strcmp(edges,'reflect1') ~= 1)
    warning('Using REFLECT1 edge-handling (use MEX code for other options).');
  end
end

if (exist('step') ~= 1)
  step = [1,1];
end

if (exist('start') ~= 1)
  start = [1,1];
end

% A multiple of step
if (exist('stop') ~= 1)
  stop = step .* (floor((start-ones(size(start)))./step)+size(im))
end

if ( ceil((stop(1)+1-start(1)) / step(1)) ~= size(im,1) )
  error('Bad Y result dimension');
end
if ( ceil((stop(2)+1-start(2)) / step(2)) ~= size(im,2) )
  error('Bad X result dimension');
end

if (exist('res') ~= 1)
  res = zeros(stop-start+1);
end

%------------------------------------------------------------

tmp = zeros(size(res));
tmp(start(1):step(1):stop(1),start(2):step(2):stop(2)) = im;

result = rconv2(tmp,filt) + res;
```

```
% V = VAR2(MTX,MEAN)
%
% Sample variance of a matrix.
%  Passing MEAN (optional) makes the calculation faster.

function res = var2(mtx, mn)

if (exist('mn') ~= 1)
  mn =  mean2(mtx);
end

if (isreal(mtx))
  res = sum(sum(abs(mtx-mn).^2)) / max((prod(size(mtx)) - 1),1);
else
  res = sum(sum(real(mtx-mn).^2)) + i*sum(sum(imag(mtx-mn).^2));
  res = res  / max((prod(size(mtx)) - 1),1);
end
```



```
% [VEC] = columnize(MTX)
%
% Pack elements of MTX into a column vector.  Just provides a
% function-call notatoin for the operation MTX(:)

function vec = columnize(mtx)

vec = mtx(:);
```

```
% RES = wpyrBand(PYR, INDICES, LEVEL, BAND)
%
% Access a subband from a separable QMF/wavelet pyramid.
%
% LEVEL (optional, default=1) indicates the scale (finest = 1,
% coarsest = wpyrHt(INDICES)).
%
% BAND (optional, default=1) indicates which subband (1=horizontal,
% 2=vertical, 3=diagonal).

% Eero Simoncelli, 6/96.

function im =  wpyrBand(pyr,pind,level,band)

if (exist('level') ~= 1)
  level = 1;
end

if (exist('band') ~= 1)
  band = 1;
end

if ((pind(1,1) == 1) | (pind(1,2) ==1))
  nbands = 1;
else
  nbands = 3;
end

if ((band > nbands) | (band < 1))
  error(sprintf('Bad band number (%d) should be in range [1,%d].', band, nbands)
);
end

maxLev = wpyrHt(pind);
if ((level > maxLev) | (level < 1))
  error(sprintf('Bad level number (%d), should be in range [1,%d].', level, maxL
ev));
end

band = band + nbands*(level-1);
im = pyrBand(pyr,pind,band);
```



```
% [HEIGHT] = wpyrHt(INDICES)
%
% Compute height of separable QMF/wavelet pyramid with given index matrix.

% Eero Simoncelli, 6/96.

function [ht] =  wpyrHt(pind)

if ((pind(1,1) == 1) | (pind(1,2) ==1))
        nbands = 1;
else
        nbands = 3;
end

ht = (size(pind,1)-1)/nbands;
```

```
% [LEV,IND] = wpyrLev(PYR,INDICES,LEVEL)
%
% Access a level from a separable QMF/wavelet pyramid.
% Return as an SxB matrix, B = number of bands, S = total size of a band.
% Also returns an Bx2 matrix containing dimensions of the subbands.

% Eero Simoncelli, 6/96.

function [lev,ind] =  wpyrLev(pyr,pind,level)

if ((pind(1,1) == 1) | (pind(1,2) ==1))
  nbands = 1;
else
  nbands = 3;
end

if ((level > wpyrHt(pind)) | (level < 1))
  error(sprintf('Level number must be in the range [1, %d].', wpyrHt(pind)));
end

firstband = 1 + nbands*(level-1)
firstind = 1;
for l=1:firstband-1
  firstind = firstind + prod(pind(l,:));
end


ind = pind(firstband:firstband+nbands-1,:);
lev  = pyr(firstind:firstind+sum(prod(ind'))-1);
```



```
% RES = ZCONV2(MTX1, MTX2, CTR)
%
% Convolution of two matrices, with boundaries handled as if the larger mtx
% lies in a sea of zeros. Result will be of size of LARGER vector.
%
% The origin of the smaller matrix is assumed to be its center.
% For even dimensions, the origin is determined by the CTR (optional)
% argument:
%      CTR   origin
%       0     DIM/2      (default)
%       1     (DIM/2)+1  (behaves like conv2(mtx1,mtx2,'same'))

% Eero Simoncelli, 2/97.

function c = zconv2(a,b,ctr)

if (exist('ctr') ~= 1)
  ctr = 0;
end

if (( size(a,1) >= size(b,1) ) & ( size(a,2) >= size(b,2) ))
    large = a; small = b;
elseif  (( size(a,1) <= size(b,1) ) & ( size(a,2) <= size(b,2) ))
    large = b; small = a;
else
  error('one arg must be larger than the other in both dimensions!');
end

ly = size(large,1);
lx = size(large,2);
sy = size(small,1);
sx = size(small,2);

%% These values are the index of the small mtx that falls on the
%% border pixel of the large matrix when computing the first
%% convolution response sample:
sy2 = floor((sy+ctr+1)/2);
sx2 = floor((sx+ctr+1)/2);

clarge = conv2(large,small);
c = clarge(sy2:ly+sy2-1, sx2:lx+sx2-1);
```