EECS C145B/ BioE C165: Lecture 6 (revised)

2D Fourier transforms in Matlab

Images of sinusoids

Make a 32×32 image of the rasied 2D sinusoid $\sin(2\pi(3x+5y))+1$:

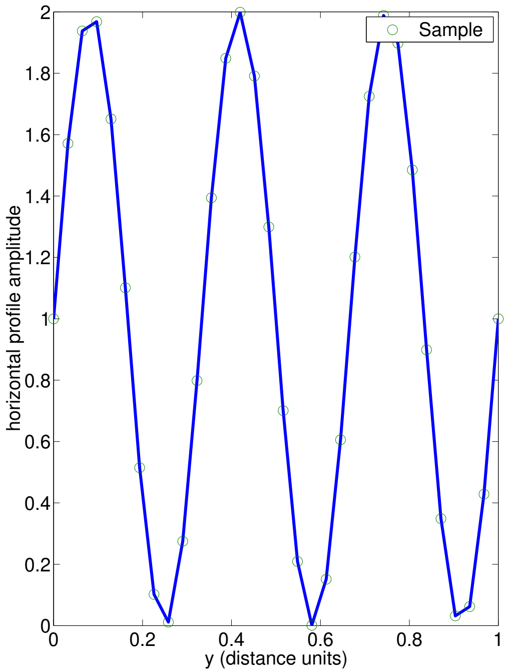
```
N = 32; % image dimension
M = N;

x = linspace(0,1,N); % axes on which sinusoid will be generated
y = fliplr(linspace(0,1,M)) % y axis starts at bottom corner;

u0 = 5; % cycles / unit (horiz. frequency)
v0 = 3; % cycles / unit (vert. frequency)

[X,Y] = meshgrid(x,y); % make Cartesian coordinate
                       % grid from axes

intensity2D = sin(2*pi*(u0*X+v0*Y)) + 1;

figure(1)
clf % clear figure
set(gca, 'fontsize', 19) % set the font size for this figure

% let's see the profile along the 1st column
h = plot(y,intensity2D(:,1),y,intensity2D(:,1),'o');

set(h(1), 'linewidth', 3); % make lines of plot thicker
set(h(2), 'markersize', 10); % make "o" markers bigger
xlabel('y (distance units)');
ylabel('horizontal profile amplitude')
legend(h(2), 'Sample')
```


Images of sinusoids

```
print -f1 -dps2 sin2ddemoprofile.ps
                              % make monochrome print file
                              % of Figure 1 for PostScript
                              % printer
orient tall
print -f1 -depsc2 sin2ddemoprofile.eps

figure(2)
clf
set(gca, 'fontsize', 17)
imagesc(x,y,intensity2D);
title(['Image of sinusoid of frequency u_0= ' num2str(u0) ...
       ' v_0= ' num2str(v0)]);
xlabel('x (distance units)');
ylabel('y (distance units)');

orient tall; % make figure tall
             % other options: portrait, landscape
axis xy % put origin at lower left

print -f2 -depsc2 sin2ddemo.eps % make color encapsulated
                                % PostScript graphic
                                % of Figure 2
                                % (To include in document)
```

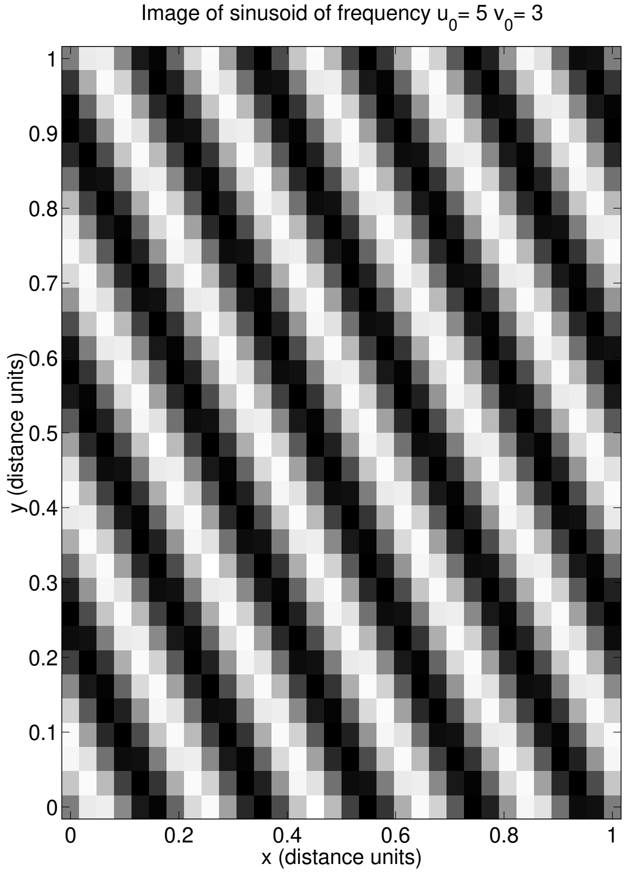
Images of sinusoids

Resulting plots



Images of sinusoids

Resulting plots



Images of sinusoids

Let's look at the coordinate system that "meshgrid" made:

```
>> X([1:3 end-1 end], [1:3 end-1 end])

ans =

         0    0.0323    0.0645    0.9677    1.0000
         0    0.0323    0.0645    0.9677    1.0000
         0    0.0323    0.0645    0.9677    1.0000
         0    0.0323    0.0645    0.9677    1.0000
         0    0.0323    0.0645    0.9677    1.0000

>> Y([1:3 end-1 end], [1:3 end-1 end])

ans =

    1.0000    1.0000    1.0000    1.0000    1.0000
    0.9677    0.9677    0.9677    0.9677    0.9677
    0.9355    0.9355    0.9355    0.9355    0.9355
    0.0323    0.0323    0.0323    0.0323    0.0323
         0         0         0         0         0
```

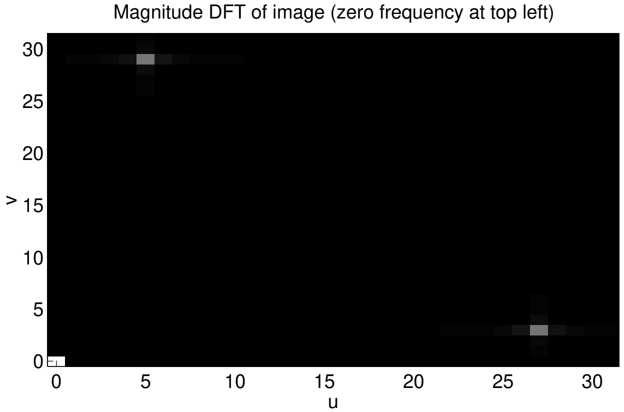
"meshgrid" and "ndgrid" are very useful for creating and plotting sampled multidimensional functions

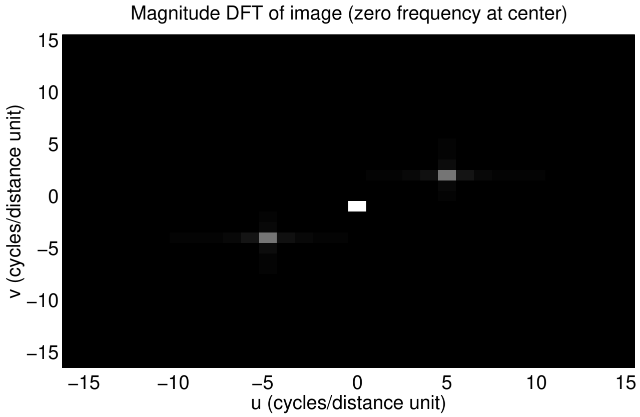
DFTs of images of sinusoids

```
dftIntensity = fftn(intensity2D); % take 2D fft
dftMagnitude = abs(dftIntensity); % calculate magnitude DFT
dftMagnitudeCentered = fftshift(dftMagnitude); % put DC in center

u_s = N; % samples / distance unit
v_s = N; % image has square pixels

uAxPreShift = linspace(0, u_s-u_s/N, N);
vAxPreShift = uAxPreShift;

figure(3)
clf
subplot(2,1,1);
set(gca, 'fontsize', 15) % set the font size for this figure
imagesc(uAxPreShift, vAxPreShift, dftMagnitude);
title(['Magnitude DFT of image ' ...
        '(zero frequency at top left)']);
xlabel('u')
ylabel('v')
axis xy

uAx = linspace(-u_s/2, u_s/2-u_s/N, N);
vAx = fliplr(uAx);

subplot(2,1,2);
set(gca, 'fontsize', 15) % set the font size for this figure
imagesc(uAx,vAx, dftMagnitudeCentered);
title(['Magnitude DFT of image ' ...
       '(zero frequency at center)']);
xlabel('u (cycles/distance unit)');
ylabel('v (cycles/distance unit)');
axis xy
orient tall
print -depsc2 -f3 sin2ddftdemo.eps
```

Images of sinusoids and their transforms

Resulting plots





DFT Tricks: Zero padding

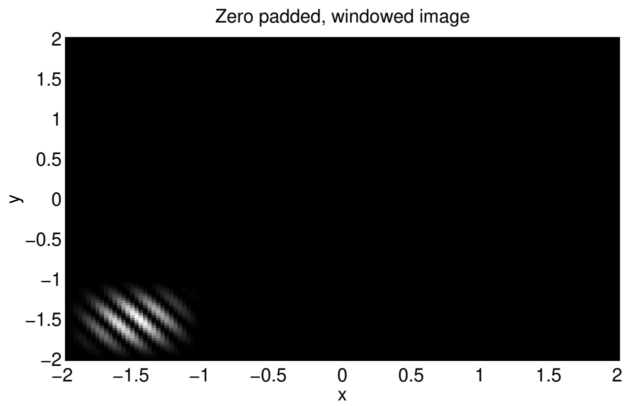
- Because the DFT is always the same size as the image, the resolution of the image limits the resolution of the spectrum.
- By appending zeros at any edge of an image, the resolution of the spectrum can be improved.

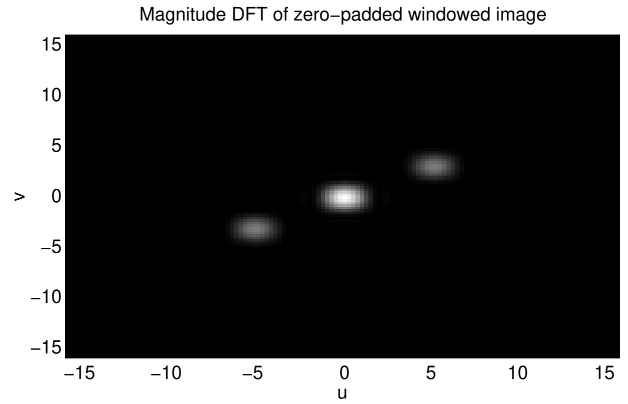
```
% apply von Hahn window to image to reduce
% image edge effects

intensity2DWin =  intensity2D .* (hanning(N)*hanning(N).');

% increase size of image to
% 128 x 128 by placing image
% in lower left corner of
% 128x128 array of zeros

padFactor = 4;
newSize = N*padFactor;
im2DPad = zeros(newSize, newSize);

% copy image of windowed sinusoid to lower left corner (origin)

im2DPad(newSize-N+1:end, 1:N) = intensity2DWin;


xAxPad = linspace(-newSize/N/2, newSize/N/2, newSize);
yAxPad = fliplr(xAxPad); % make y axis increase from
                         % lower left corner up
```

Windowing and zeropadding

```
figure(4)
clf
subplot(2,1,1);
set(gca, 'fontsize', 15)
imagesc(xAxPad, yAxPad, im2DPad)
colormap(gray)
axis xy

title('Zero padded, windowed image')
xlabel('x')
ylabel('y')

% u_s and v_s do not change. The
% DFT sample spacing does though
uAxPad = linspace(-u_s/2, u_s/2-u_s/(N*padFactor), ...
       N*padFactor);
vAxPad = fliplr(uAxPad);

dftPad = fftn(im2DPad);
dftMagnitudePad = abs(dftPad);

subplot(2,1,2);
set(gca, 'fontsize', 15)
imagesc(uAxPad, vAxPad, fftshift(dftMagnitudePad))
colormap(gray)
axis xy
title('Magnitude DFT of zero-padded windowed image')
xlabel('u')
ylabel('v')

orient tall
print -depsc2 -f4 sin2ddftdemopadwin.eps
```


Windowing and zero padding
Resulting plots





Zero padding increases the resolution of the spectrum.

Windowing reduces edge effects but decreases spectral resolution.

Writing a Matlab function to compute symmetric DFT axis indices

The following function is useful for making plot axes for magnitude spectra and for understanding the conjugate symmetry of the DFT for even and odd length sequences.

```
% function ax = makedftaxis(lenAx, dbg)
% Make axis of indices for symmetric
% DFTs of length N points
% If dbg == 1 will check to see that
% axis is a symmetric valid DFT axis

function ax = makedftaxis(lenAx, dbg)

if nargin < 2 % If only 1 arg is given
  dbg = 0;    % set debug mode off
end

axMid = round(lenAx/2);
axStart = 0:axMid-mod(lenAx,2);
axEnd = fliplr(axStart(2:(end-1+mod(lenAx,2))));
ax = [axStart axEnd];


if dbg
  F = fft(ax);
  checkreim(F); % check to see that this vector
                % looks symmetric to the DFT

end
```


Writing a Matlab function to compute symmetric DFT axis indices

The function can then be invoked like any other Matlab function:

```
>> makedftaxis(8)
ans =

     0     1     2     3     4     3     2     1

>> makedftaxis(7)
ans =

     0     1     2     3     3     2     1

>> fftshift(makedftaxis(7))
ans =

     3     2     1     0     1     2     3

>> fftshift(makedftaxis(8))
ans =

     4     3     2     1     0     1     2     3

>> ifft(makedftaxis(7))
ans =

  Columns 1 through 4

1.7143  -0.7213 + 0.0000i  -0.0440 + 0.0000i  -0.0919 + 0.0000i

  Columns 5 through 7

  -0.0919 + 0.0000i  -0.0440 + 0.0000i  -0.7213 + 0.0000i
```

The last result shows that the vector looks symmetric to the DFT (it has a real inverse transform).

Writing a Matlab function to check that a matrix has only a small imaginary part

The following function is useful to check if it is safe to remove the imaginary part of a matrix that you expect to be real (Matlab will always return a complex matrix even if the imaginary components are very small. See last result on previous page).

```
% function realMat = checkreim(mat, tolFrac, terminal)
% Check matrix to see if the max of the abs of the imag part
% exceeds tolFrac * the mean of the abs of the real part.
% If matrix is sufficiently real, return real matrix.
% If terminal == 0, return empty matrix if mat is complex.
% If terminal == 1, stop if mat is complex.

function realMat = checkreim(mat, tolFrac, terminal)

if nargin < 2
  tolFrac = 1e-6;
end

if nargin < 3
  terminal = 1; % stop on imaginary matrix
end

mxr = mean(abs(real(mat(:))));
mxi = max(abs(imag(mat(:))));


if (mxi / mxr) > tolFrac
  if terminal
    error('checkreim(): input matrix is appreciably complex');
  else
    realMat = [];
  end
else
  realMat = real(mat);
end
```


Writing a Matlab function to check that a matrix has only a small imaginary part

```
>> checkreim(ifft(makedftaxis(7)))

ans =

1.7143  -0.7213  -0.0440  -0.0919  -0.0919  -0.0440  -0.7213

>> checkreim(ifft(makedftaxis(7))+0.001i)
??? Error using ==> checkreim
checkreim(): input matrix is appreciably complex

>> tol = 0.1; % we will tolerate an imaginary part
              % having imaginary elements whos
              % absolute values are less than 1/10th
              % of the mean absolute value of the
              % real part
>> checkreim(ifft(makedftaxis(7))+0.001i, 0.1)

ans =

1.7143  -0.7213  -0.0440  -0.0919  -0.0919  -0.0440  -0.7213

>> terminate = 0;
>> checkreim(ifft(makedftaxis(7))+1i, 0.1, terminate)

ans =

     []
```

Implementing 2D frequency domain filters: the ideal lowpass filter

```
% read in jpeg image

im = imread('oakland.jpg');
[rows,cols,colors] = size(im);

% make axes for image. Define image
% width as 1 unit
xDim = cols;
yDim = rows;
xAx = linspace(-0.5, 0.5, xDim);
% need to have origin at lower left
yAx = fliplr(linspace(-0.5, 0.5, yDim)*yDim/xDim);

% make frequency space axes
uDim = cols;
vDim = rows;

% general axis form for odd and even length axes
uAx = linspace(-floor(uDim/2), round(uDim/2-1), uDim);

% assuming square pixels, our max frequency sampled
% in y direction is same as in x, but we
% have larger spacing between DFT samples
vAx = fliplr(linspace(-floor(uDim/2), round(uDim/2-1), vDim));

% make Cartesian grid with origin at lower left
[U,V] = meshgrid(uAx, vAx);

% set normalized cutoff frequency, assuming sampling freq
% is normalized to 1
uCutNorm = 0.020;
vCutNorm = 0.45;
```


Implementing 2D frequency domain filters: the ideal lowpass filter

```
us = uDim; % assume width of image is 1 unit
vs = us;
uCut = uCutNorm*us;
vCut = vCutNorm*vs;

% get matrix indices for all the passband elements of H
passBandHInd = find(abs(U) <= uCut & abs(V) <= vCut);

HShift = zeros(size(U));
HShift(passBandHInd) = 1;

% move HShift so that zero frequency is at top left
HProto = fftshift(HShift);
% due to numerical errors, HProto is not perfectly
% conjugate symmetric. This trick forces this.
% It is important to check that the filter
% returned is close to what you want.

h = ifftn(HProto);
hr = real(h);
H = fftn(hr);

% perform filtering on each color component
% separately

gr = ifftn(H.*fftn(im(:,:,1)));
gg = ifftn(H.*fftn(im(:,:,2)));
gb = ifftn(H.*fftn(im(:,:,3)));

g(:,:,1) = checkreim(gr);
g(:,:,2) = checkreim(gg);
g(:,:,3) = checkreim(gb);
```

Implementing 2D frequency domain filters: the ideal lowpass filter

```
% imagesc needs color images scaled
% between 0 and 1, so enforce
% this

gScaled = (g-min(g(:)));
gScaled = gScaled ./ max(gScaled(:)) * 1;

% plot original image
figure(1)
set(gca,'fontsize',20)
imagesc(xAx, yAx, im);
axis xy
xlabel('x (units of image width)')
ylabel('y (units of image width)')
colormap(gray)
title('Image f[m,n]')
orient portrait

% plot LPF
figure(2)
set(gca,'fontsize',20)
imagesc(uAx, vAx, HShift);
axis xy
colorbar
xlabel('u (cycles/image width)')
ylabel('v (cycles/image width)')
colormap(gray)
title('Low pass filter H[k,l]')
orient portrait
```


Implementing 2D frequency domain filters: the ideal lowpass filter

```
% plot spectrum for blue channel
FBlueLogMagn = fftshift(log(abs((fftn(im(:,:,3))))));
figure(3)
clf
set(gca,'fontsize',20)
imagesc(uAx, vAx, FBlueLogMagn)
axis xy
colorbar
xlabel('u (cycles/image width)')
ylabel('v (cycles/image width)')
colormap(gray)
title('Log magnitude spectrum of image F[k,l] (blue)')
orient portrait

figure(4)
clf
set(gca,'fontsize',20)
imagesc(xAx, yAx, gScaled);
axis xy
axis normal
xlabel('x (units of image width)')
ylabel('y (units of image width)')
colormap(gray)
title('Filtered image g[m,n]')
orient portrait
```

Implementing 2D frequency domain filters: the ideal lowpass filter

```
figure(5)
clf
set(gca,'fontsize',18)
% plot only central region
xSel = round(xDim/3:2*xDim/3);
ySel = round(yDim/3:2*yDim/3);
hrShift = ifftshift(hr);
mesh(xAx(xSel), yAx(ySel), ...
     hrShift(ySel,xSel));
axis xy
axis normal
xlabel('x (units of image width)')
ylabel('y')
colormap(pink)
title('PSF h[m,n]')
orient portrait
xl = xlim;
yl = ylim;
print -f5 -depsc2 ../../eps/fourierfilt_h1.eps
zoom(4) % zoom im 4 times
view(-37.5,60) % view from different angle

print -f1 -depsc2 ../../eps/fourierfilt_f.eps
print -f2 -depsc2 ../../eps/fourierfilt_H.eps
print -f3 -depsc2 ../../eps/fourierfilt_F.eps
print -f4 -depsc2 ../../eps/fourierfilt_g.eps
print -f5 -depsc2 ../../eps/fourierfilt_h2.eps
```


Implementing 2D frequency domain filters: The Butterworth low-pass filter

The ancillary code needed to implement the Butterworth filter is the same as for the ideal low-pass filter. The specific code for generating $H[k, l]$ is shown below:

```
% set normalized cutoff frequency, assuming sampling freq
% is normalized to 1
rhoCutNorm = 0.025;
n = 3; % order of the filter

us = uDim; % assume width of image is 1 unit
rhoCut = rhoCutNorm*us;

% Implement Butterworth filter
Rho = sqrt(U.^2+V.^2); % radial coordinate system
rhoc = 1/2;
HShift = 1./(1+(Rho/rhoCut).^(2*n));
```

Implementing 2D frequency domain filters: The Gaussian high-pass filter

The specific code for generating $H[k,l]$ is shown below:

```
% set normalized cutoff frequency, assuming sampling freq
% is normalized to 1
rhoCutNorm = 0.05;

us = uDim; % assume width of image is 1 unit
rhoCut = rhoCutNorm*us;

Rho = sqrt(U.^2+V.^2);

% Implement Gaussian high-pass filter
HShift = 1 - exp(-Rho.^2/(2*rhoCut^2));
```


Implementing 2D frequency domain filters: The Butterworth bandstop filter

This is the code that added contamination to the image:

```
% read in jpeg image

im = imread('oakland.jpg');
[rows,cols,colors] = size(im);

% make axes for image. Define image
% width as 1 unit

xDim = cols;
yDim = rows;
xAx = linspace(-0.5, 0.5, xDim);
yAx = fliplr(linspace(-0.5, 0.5, yDim)*yDim/xDim);
[X,Y] = meshgrid(xAx,yAx);
f0 = 80;
cosNoise = 90*(1+0.5*cos(2*pi*f0*(X+Y)));

% im is of type uint8 (unsigned 8 bit integer)
% Most Matlab operators work only with real numbers
% so we convert to double precision floating point.

im= double(im) + repmat(cosNoise, [1 1 3]);
im = (im-min(im(:)))/max(im(:));
```

Implementing 2D frequency domain filters: The Butterworth bandstop filter

The code that generates $H[k,l]$ is shown below:

```
% f0 is x and y frequency of contaminating sinusoid
rhoCutL = f0*sqrt(2)-40;
rhoCutH = f0*sqrt(2)+40;

% Implement Butterworth bandstop filter
Rho = sqrt(U.^2+V.^2);
n=10; % filter order

% low cut-off low-pass filter
HShiftL = 1./(1+(Rho/rhoCutL).^(2*n));
% high cut-off low-pass filter
HShiftH = 1./(1+(Rho/rhoCutH).^(2*n));

HShift = 1 - HShiftH +HShiftL;
```


Reconstruction fitler 1D demo: Code example

This code generated the 1D sinc interpolation demo.

```
% set the number of samples in our discrete signal
N=10;
tEnd = 1; % end time
t = linspace(0,tEnd,N); % time axis
% our signal with random normal component
sig = cos(2*pi*t*3) + ...
      1/2 *cos(2*pi*t*2+pi/4).*cos(2*pi*t*8+pi/9)+randn(1,N);

figure(1)
set(gca, 'fontsize', 22)
hs =stem(t,sig)
set(hs, 'markersize', 10)
set(hs, 'linewidth', 3)
title('Discrete audio signal')
xlabel('t (seconds)')
ylabel('f[n]')
xlim([0-0.1 tEnd+0.1])
orient portrait
y = sig;

figure(2)
clf
set(gca, 'fontsize', 22)
hs =stem(t,sig) % needed because arrow.m has bug
set(hs, 'markersize', 0.1)
for i = 1:length(y)
    ha = arrow([t(i) 0], [t(i) y(i)], ...
    'linewidth',2,'Tipangle',30,'width',2)
end
title('Discrete audio signal converted to analog signal')
xlabel('t (seconds)')
ylabel('f_a(t) (volts)')
```

Reconstruction fitler 1D demo: Code example

```
xlim([0-0.1 tEnd+0.1])
orient portrait

T = t(2)-t(1); % get sample period
% make mock continuous axis
tCont = linspace(0,tEnd,N*100);

figure(3)
clf
set(gca, 'fontsize', 22)
hold on % stop new plots from erasing old ones
interpFnSum = zeros(1,length(tCont)); % zero convolution sum
for i = 1:length(y)
  interpFn = y(i)*sinc((tCont-t(i))/T);
  interpFnSum = interpFnSum + interpFn; % add this stamped copy
  hp(i) = plot(tCont, interpFn)
end
set(hp, 'linewidth', 2)
hold off
title('f_a(t) "stamped" with sinc(t-n/T) before summing')
xlabel('t (seconds)')
```


Reconstruction fitler 1D demo: Code example

```
ylabel('f_q(t)')
xlim([0-0.1 tEnd+0.1])
orient tall

figure(4)
clf
set(gca, 'fontsize', 22)
hp = plot(tCont, interpFnSum)
set(hp, 'linewidth', 2)
hold off
title('f_r(t), the reconstructed continuous function')
xlabel('t (seconds)')
ylabel('f_r(t)')
xlim([0-0.1 tEnd+0.1])
orient tall

print -f1 -depsc2 ../../eps/dac1dfdisc.eps
print -f2 -depsc2 ../../eps/dac1dfcont.eps
print -f3 -depsc2 ../../eps/dac1dfsinc.eps
print -f4 -depsc2 ../../eps/dac1dfrecon.eps
```