Subject: FFT: How to select BW and frequency
Posted by hahn on Mon, 21 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Hi,

I shot several photos of objects that were behind a curtain. Now I want to eliminate the vertical and horizontal white streaks from the pictures. The photos are stored as 24 bit tiff images. To get familiar with the problem I read the blue image of one picture into the FFT demo of IDL 3.01 and played with a band pass filter to get the vertical lines of the curtain as good as possible, having the band width as low as possible.

I manage to see the vertical lines in the filtered picture but when I switch to the "band reject" function of the FFT demo, the lines are not removed.

When I look at the first scan line of the photo I find 7 pixels that are o.k. and 2 pixels that represent a part of the curtain. So the following question comes up:

What is the relation of the number of pixels I want to change and the bandwidth and frequency in the FFT program?

I selected the FFT demo as a first iteration because the streaks made by the curtain are highly periodic. But, what other program can be used instead of FFT?

Thanks for any help,

Norbert

Subject: Re: FFT: How to select BW and frequency
Posted by Peter Mason on Wed, 23 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

On Mon, 21 Jul 1997, Norbert Hahn wrote:

> I shot several photos of objects that were behind a curtain. Now I want
> to eliminate the vertical and horizontal white streaks from the pictures.
> The photos are stored as 24 bit tiff images. To get familiar with the
> problem I read the blue image of one picture into the FFT demo of
> IDL 3.01 and played with a band pass filter to get the vertical lines
> of the curtain as good as possible, having the band width as low
> as possible.
>

> I manage to see the vertical lines in the filtered picture but when I
> switch to the "band reject" function of the FFT demo, the lines
> are not removed.
>
> When I look at the first scan line of the photo I find 7 pixels that
> are o.k. and 2 pixels that represent a part of the curtain. So the
> following question comes up:
>
> What is the relation of the number of pixels I want to change and
> the bandwidth and frequency in the FFT program?
>
> I selected the FFT demo as a first iteration because the streaks
> made by the curtain are highly periodic. But, what other program can
> be used instead of FFT?

I think that you could use an FFT for this, but that a "discrete cosine transform" would be better.

If the stripes in the image are nicely vertical and periodic then I guess you could calculate where they should be in the FFT'd image, but I've never done this before and couldn't advise you on it.
I'd be inclined just to manually identify and zero offending areas in the FFT'd image rather than attempting anything sophisticated like a bandpass/ reject filter.

You could also try a principal components (or MNF) transform on the image before the FFT. (This might just concentrate the noise in 1 or 2 bands.)

This kind of noise usually sticks out like dogs' ... in a FFT image - as very obvious lines or spots. (I think that you're meant to get spots in the FFT if the lines in the original are well-defined, straight, strictly horizontal/ vertical, and periodic. More "natrural" lines in the original will smear the spots in the FFT somewhat.)
Once you have the FFT "power spectrum" image displayed, you can get rid of much of the noise by finding the positions of these spots and lines and manually zeroing them in the FFT array (which is complex). (e.g., You zero a thin/ small rectangle of pixels around the noise. It can get a bit tricky if the noise intrudes close to the zero frequency - where most of the desired data usually is.)

One problem with working on an FFT of a real image is that the FFT gives you more data than you want. It has real and imaginary components. If you zap anything in one component, you must also zap it (symmetrically) in the other, otherwise you'll most likely get a complex image when you transform back.
You also have to deal with the way FFT routines organise the transform array - the position of the zero frequency can seem strange.

The discrete cosine transform (DCT) is much easier to work with, and is also a little superior because it has less "edge effects".
A DCT is just a special case of an FFT: if you take an FFT of a function that is symmetrical about 0 then you get one in which all the SIN terms (the imaginary component) are zero. This is essentially a "cosine transform".
In practice, you can get a DCT of any real function by first "doubling up" the function to make it symmetrical, and then taking its FFT. (This doubling up is also the reason why the DCT has less edge effects than the FFT of the undoubled function.) You then use only the real component of this FFT.

Here's some code to get the forward or reverse DCT of a 2D image. (It may not be the "proper way" to do a DCT - there are various - but it works.)
It positions the zero frequency at top left.

```
 ;============================================================ ======
function DCT_1D,arr,direc,w,eel,oel
; Sort-of muckaround 1D Discrete Cosine Transform (but real and reversible)
; ARR           one-dimensional array on which to calculate the transform
; DIREC          -1 == forward transform, 1 == reverse transform.
; W,EEL,OEL     fixup / resampling stuff set up by caller
n=n_elements(arr) & b=reform(arr,n)
;eel=lindgen(long(n-1)/2+1)*2 & oel=rotate((eel+1),2)
;if ((n mod 2) ne 0) then oel=oel(1:n_elements(oel)-1)
if (direc lt 0) then begin
  ;w=2.0*exp((cindgen(n)*complex(0,-1)*!pi)/(2.0*n))
  b=complex(b([eel,oel]))
  b=fft(b,-1,/overwrite)
  b(0)=b(0)*0.5
  return,float(b*w)
endif else begin
  b(0)=b(0)*2.0
  ;w=0.5*exp((cindgen(n)*complex(0,1)*!pi)/(2.0*n))
  c=[0.0,b(n-1L-lindgen(n-1L))] & b=complex(b,-c)*w
  b=fft(b,1,/overwrite)
  j=fltarr(n) & j([eel,oel])=float(b)
  return,j
endelse
end
 ;============================================================ ======
pro DCT_2D_FORWARD,rawd,trfd
; Steam-driven 2D forward Discrete Cosine Transform
; RAWD  2D real image to be transformed (input)
; TRFD  2D real transformed image (output)
  j=size(rawd)
  if (j(0) ne 2) then message,'Give me a 2D input image.'
  if (j(3) ne 4) and (j(3) ne 5) then message,$
    'Give me a 2d REAL or DOUBLE input image.'
```

```
  ns=j(1) &nl=j(2)
  trfd=rawd
  eel=lindgen((ns-1L)/2L+1L)*2L & oel=rotate((eel+1L),2L)
  if ((ns mod 2L) ne 0L) then oel=oel(1L:n_elements(oel)-1L)
  w=2.0*exp((cindgen(ns)*complex(0,-1)*!pi)/(2.0*ns))
  for j=0L,nl-1L do trfd(0,j)=dct_1d(trfd(*,j),-1,w,eel,oel)
  eel=lindgen((nl-1L)/2L+1L)*2L & oel=rotate((eel+1L),2L)
  if ((nl mod 2L) ne 0) then oel=oel(1L:n_elements(oel)-1L)
  w=2.0*exp((cindgen(nl)*complex(0,-1)*!pi)/(2.0*nl))
  for j=0L,ns-1L do trfd(j,0)=reform(dct_1d(trfd(j,*),-1,w,eel,oel),1L,nl)
return
end
 ;============================================================ ======
pro DCT_2D_REVERSE,trfd,utrfd
; Steam-driven 2D reverse Discrete Cosine Transform
; TRFD  2D real DCT image to be "untransformed" (input)
; UTRFD  2D real "restored" image (output)
  j=size(trfd)
  if (j(0) ne 2) then message,'Give me a 2D input image.'
  if (j(3) ne 4) and (j(3) ne 5) then message,$
    'Give me a 2d REAL or DOUBLE input image.'
  ns=j(1) &nl=j(2)
  utrfd=trfd
  eel=lindgen((nl-1L)/2L+1L)*2L & oel=rotate((eel+1L),2L)
  if ((nl mod 2L) ne 0) then oel=oel(1L:n_elements(oel)-1L)
  w=0.5*exp((cindgen(nl)*complex(0,1)*!pi)/(2.0*nl))
  for j=0L,ns-1L do utrfd(j,0)=reform(dct_1d(utrfd(j,*),1,w,eel,oel),1L,nl)
  eel=lindgen((ns-1L)/2L+1L)*2L & oel=rotate((eel+1L),2L)
  if ((ns mod 2L) ne 0L) then oel=oel(1L:n_elements(oel)-1L)
  w=0.5*exp((cindgen(ns)*complex(0,1)*!pi)/(2.0*ns))
  for j=0L,nl-1L do utrfd(0,j)=dct_1d(utrfd(*,j),1,w,eel,oel)
return
end
 ;============================================================ ======
```

Peter Mason