IDL tutorial for physicists

0011 0010 1010 1101 0001 0100 1011

Year 2005

Robert Scholten

WEB resources

Optics Group setup hints and useful links

http://optics.ph.unimelb.edu.au/help/idl/idl.html

IDL homepage

www.rsinc.com

The main guy for IDL education plus annotated links to useful IDL libraries

www.dfanning.com

A good IDL tutorial

http://scv.bu.edu/SCV/Tutorials/IDL/idl_webtut.html

Archive of comp.lang.idl-pvwave Newsgroup

http://cow.physics.wisc.edu/~craigm/idl/newsgroup.html

Help!

- Online hypertext help: ?

  Specific commands: `? plot` for help on `plot`

Executive commands

- Starting IDL
  - **`% idlde`** — **development environment**
  - **`% idl`** — **classic style**
- Executive commands
  - **`.run`** — **compile/run program**
  - **`retall`** — **reset after errors**
  - command editing: arrow keys
  - comments "**`;`**" and long lines "**`$`**"

- More advanced
  - suspend **`^Z`** and restart **`fg`** — **get out of IDL temporarily**
  - **`save, /all`** and **`restore`** — **save everything**
  - **`spawn,'command'`** — **run a Unix command**

Sample IDL session

```
baker~> idl
IDL Version 6.1.1 (linux x86 m32). (c) 2004, Research
   Systems, Inc.
Installation number: 97562-1.
Licensed for use by: University of Melbourne

IDL> a=5
IDL> b=[2,3,4]
IDL> print,a,b
       5       2       3       4
IDL> c=a*b
IDL> print,c
      10      15      20
IDL> exit
baker~>
```

Journal

- You can record everything you do in a journal file:

```
IDL> journal
IDL> ; do stuff (will go to idlsave.pro)
IDL> print,'hello'
hello
IDL> journal
IDL> @idlsave.pro
hello
IDL>
```

Main Level Programs

In your favourite editor (emacs, nano, pico) edit the `idlsave.pro` file and save it as `fred.pro`:

```
a=6
b=[1,2,3]
print,a,b
c=a*b
print,c
end
```

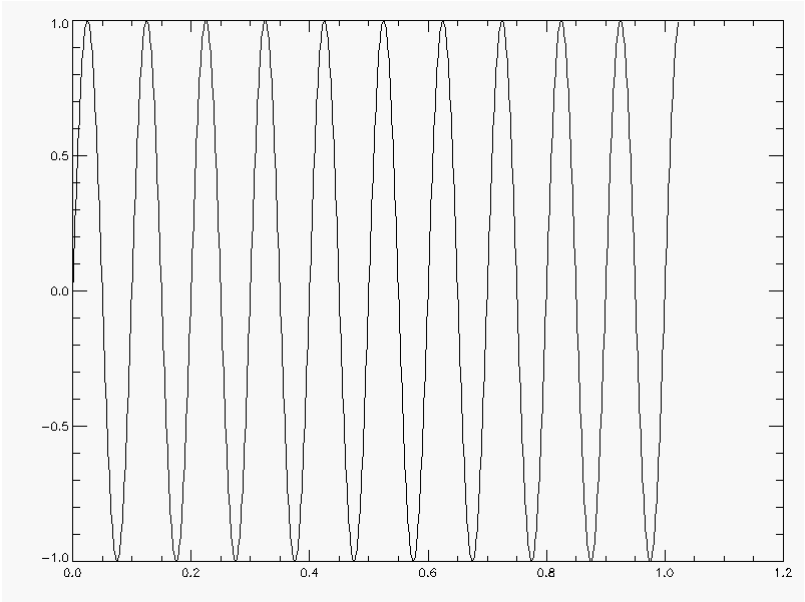
In IDL:

```
IDL> .run fred
% Compiled module: $MAIN$
       6       1       2       3
       6      12      18
```

Main level program file variables exist at the MAIN program level.

Problem 1: Signal processing

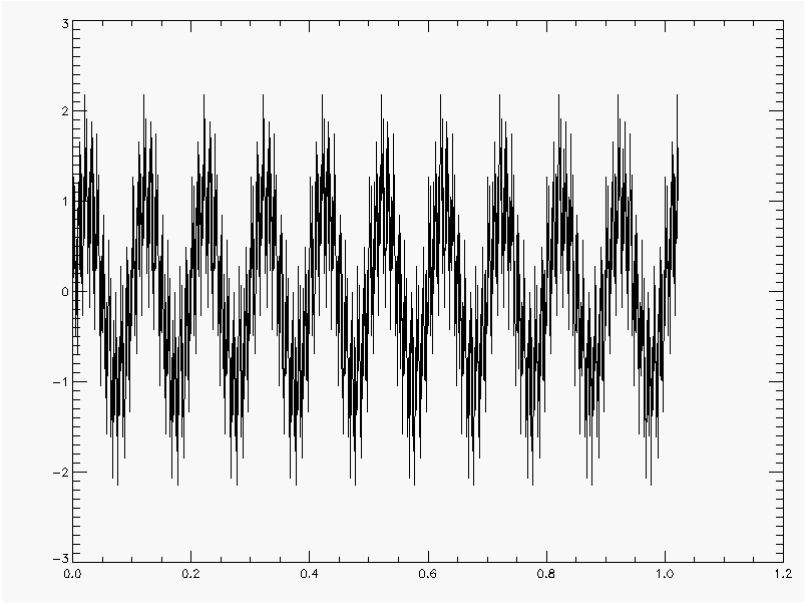
```
; fabricate some simple waveform
delt=0.001 ; time resolution
npts=1024  ; number of time data points (should be power of 2)
t=findgen(npts)*delt
f1=10*(2*!pi)    ; frequency of waves
s1=sin(f1*t)
plot,t,s1
```



http://www.msi.umn.edu/software/idl/tutorial/idl-signal.html

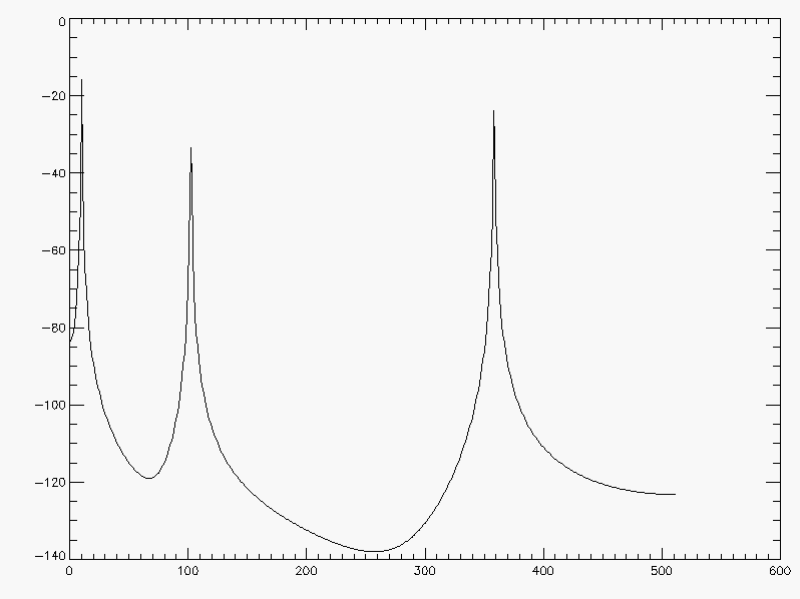
Signal processing

```
; add some more components
f2=100*(2*!pi) & s2=0.5*sin(f2*t+0.1*!pi)
f3=350*(2*!pi) & s3=0.8*sin(f3*t-0.2*!pi)
s=s1+s2+s3
plot,t,s
; save to sound file
rate=1/delt
write_wav,'tute1.wav',s*16000,rate
```



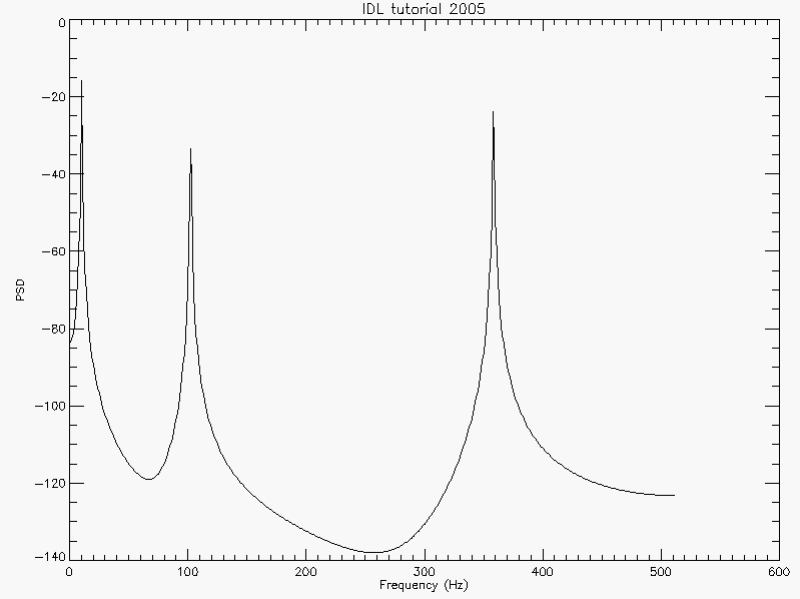
Signal processing (FFT)

```
; fourier analyse
spec=fft(s,-1)
delf=1d0/delt
f=findgen(npts/2)*delf
plot,f/1000.0,10.0*alog(abs(spec(0:npts/2-1))^2)
```



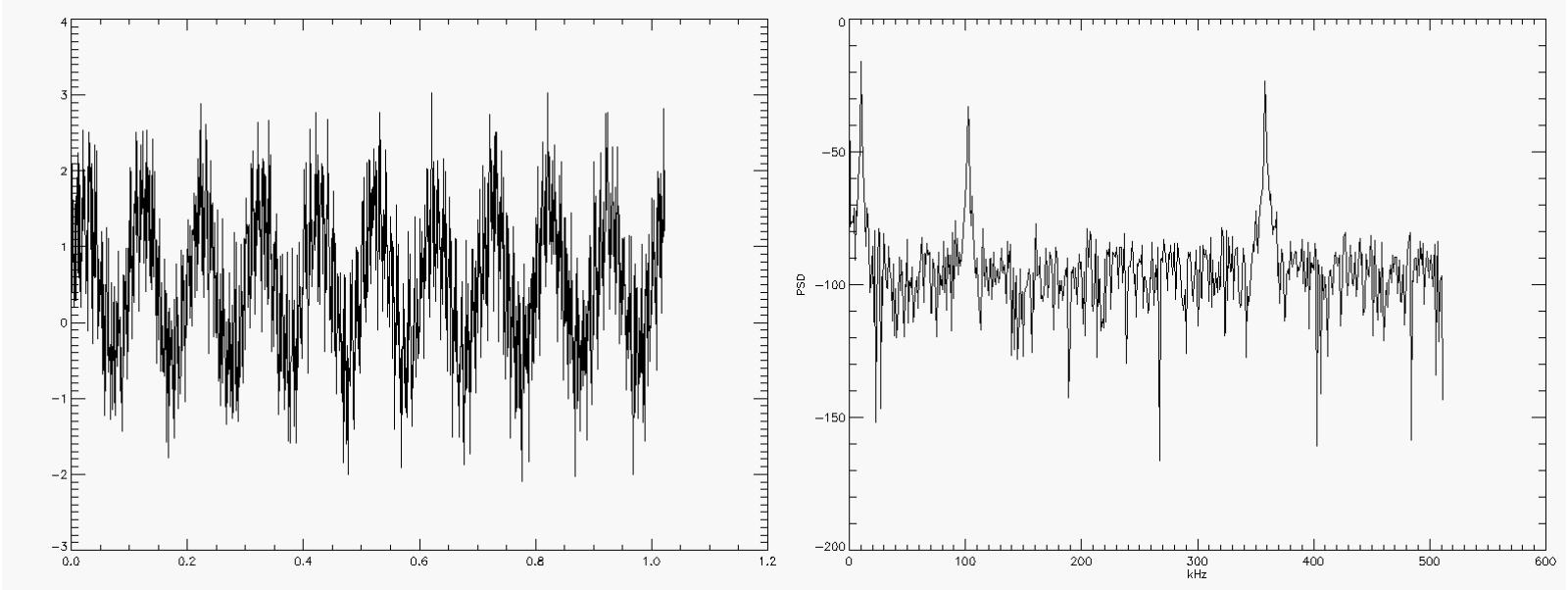
Signal processing…

```
; make plot a little prettier
plot,f/1000.0,10.0*alog(abs(spec(0:npts/2-1))^2),xtitle='Frequency
   (Hz) ',ytitle='PSD',title='IDL tutorial 2005'
```



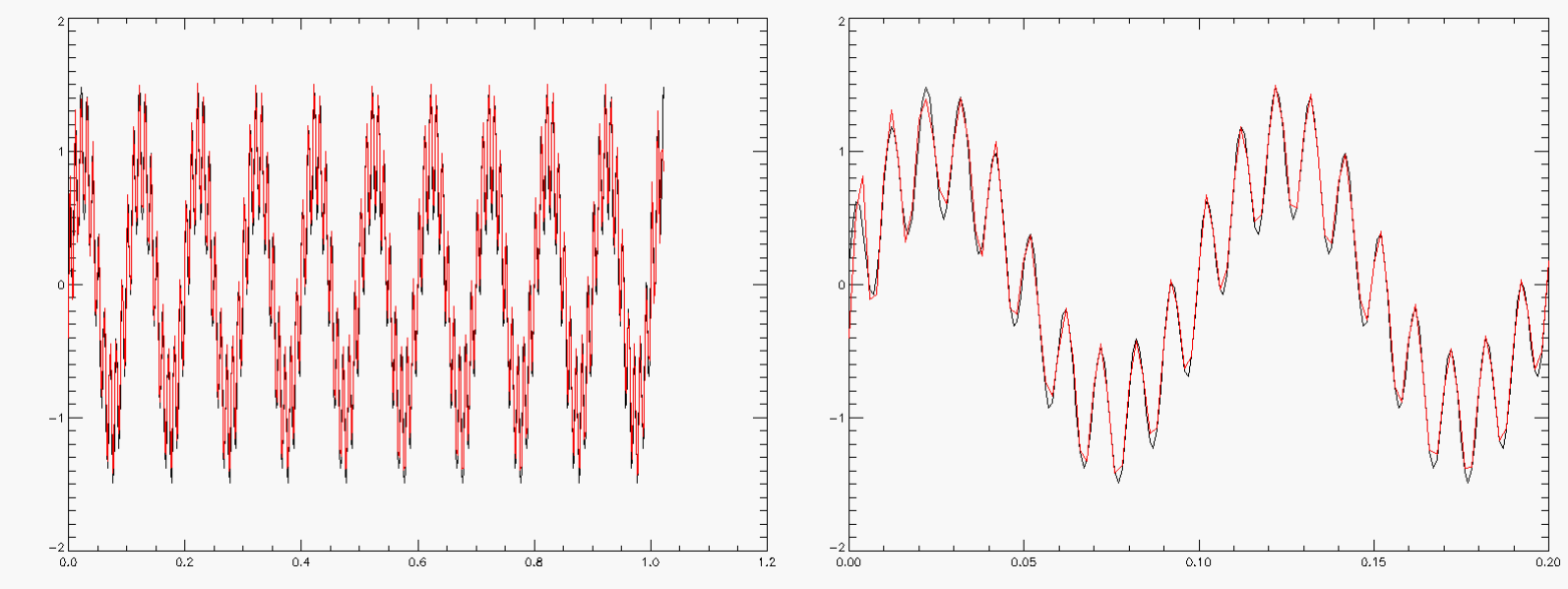
Signal processing…

```
; add some noise
noise=randomu(seed,npts)
sn=s+noise
plot,t,sn
specn=fft(sn,-1)
delf=1d0/delt
f=findgen(npts/2)*delf
plot,f/1000.0,10.0*alog(abs(specn(0:npts/2-1))^2),
   xtitle='kHz',ytitle='PSD'
```



Frequency domain filtering

```
; frequency domain filtering, 20dB suppression around f0
f0=350*delf & fsig=25*delf
filter=fltarr(npts/2) ;using only half spectrum - throw away phase
filter(*)=1.0d0
fpts=where(abs(f-f0) lt fsig)
filter(fpts)=filter(fpts)*1d-4
filtspec=spec(0:npts/2-1)*filter
sigfiltered=fft(filtspec,1)
plot,t,s1+s2
oplot,t*2,sigfiltered*2,col='0000ff'x      … ,xrange=[0,0.2]
```



Multiple plots

```
; multiple windows
window,1
window,2,xsize=400,ysize=300
wset,1  &  plot,t,s1
wset,2  &  plot,t,s2

; multiple plots on one page
!p.multi=[0,2,3] ; two columns, three rows
    plot,blah blah blah (top left)
    plot,blah blah blah (top right)
    plot,(next row, left)
    plot,…and right
    plot,etc…
    oplot,…still works…
!p.multi = 0
```

Multiple plots

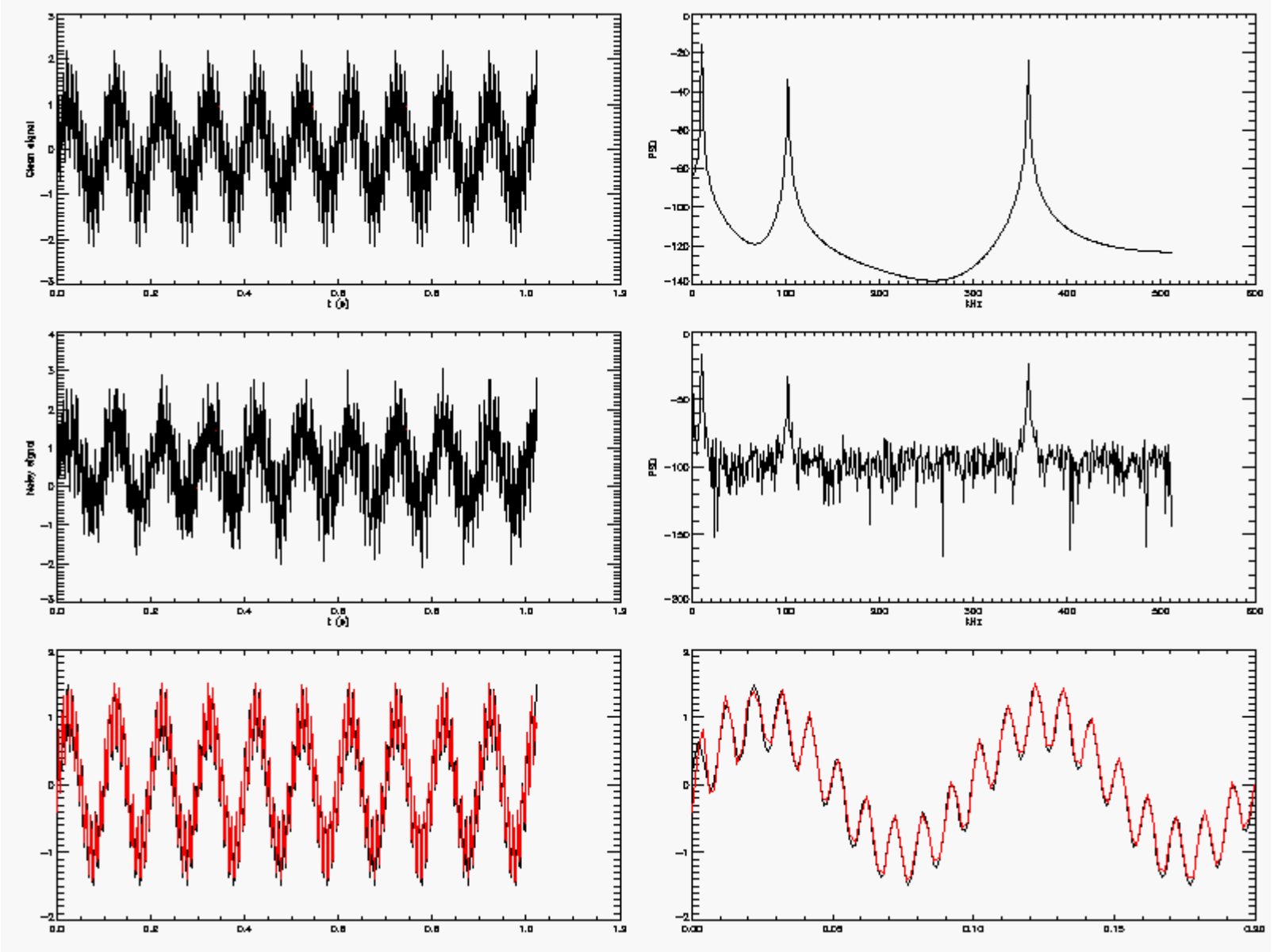


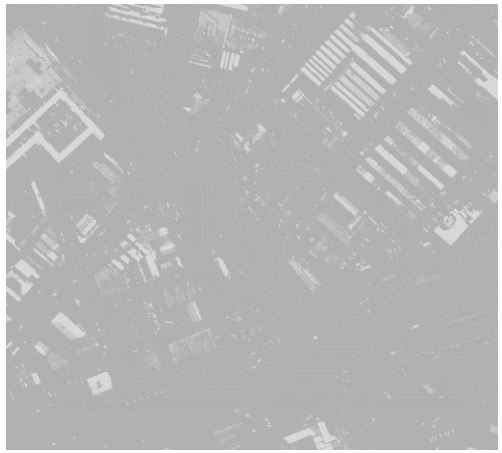
Image processing: arrays and images

```
; open an image
img=read_image('Ikonos_pan.jpg')
s=size(img) & ncol=s(1) & nrow=s(2)
tv,img*0.3
tvscl,img*0.3
```

Contrast

```
tv,img>100 ; whichever is greater
tv,img<100 ; whichever is less
tv,img>180<220
```

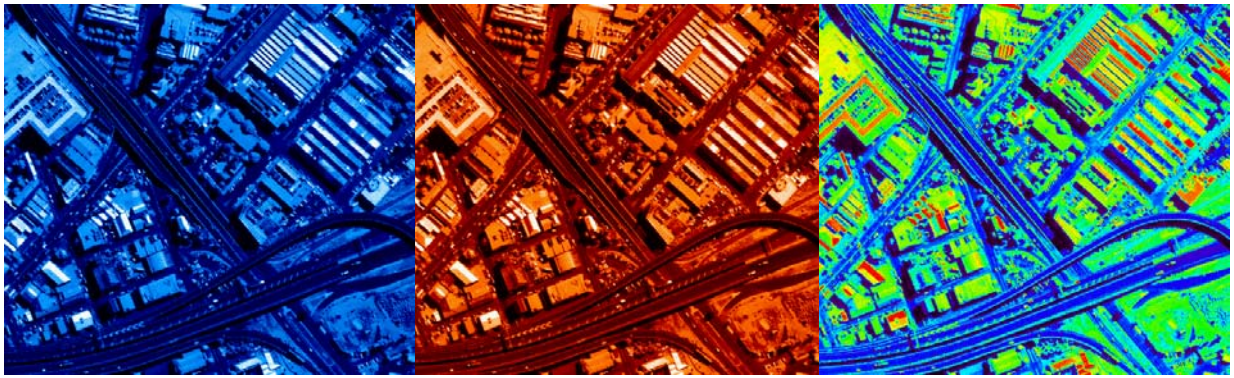
The last command shows **`img`** where pixels are between 180 and 220. Pixels are set to 180 where **`img <=180`** and set to 220 where **`img >=220`**



Colour tables

- IDL can use true colour (24-bit), or a colour lookup table
- Switch to colour lookup with:
  `device,decomposed=0`

```
device,decomposed=0       ; makes colour management easier

img=read_image('Ikonos_pan.jpg')
s=size(img)& ncol=s(1)& nrow=s(2)
wdelete & window,xsize=ncol,ysize=nrow,title='Colour mapped'

for myct=0,40 do begin
  loadct,myct
  tvscl,img
  wait,2
endfor
end
```

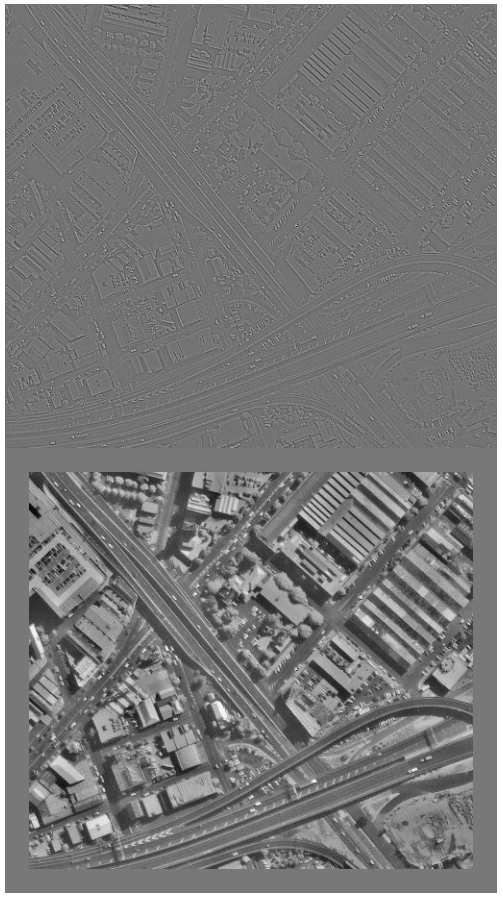


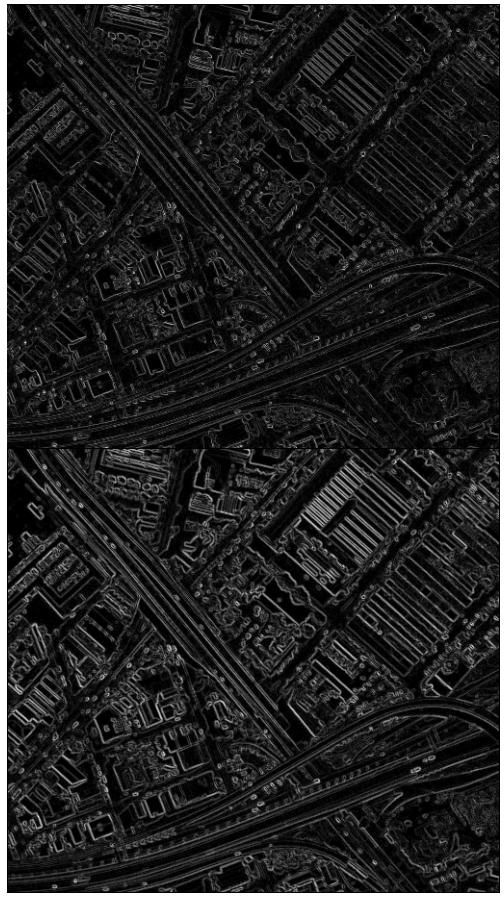
Resizing

```
; resizing
img2=rebin(img,2*ncol,2*nrow)
wdelete
window,xsize=2*ncol,ysize=2*nrow,title='rebinned'
tvscl,img2

img3=congrid(img,1.5*ncol,1.5*nrow)
wdelete
window,xsize=1.5*ncol,ysize=1.5*nrow,title='congridded'
tvscl,img3
```

Image processing

```
; smoothing
tvscl,smooth(img,3)

; unsharp mask
tvscl,float(img)-smooth(img,2)
tvscl,float(img)-smooth(img,50)

; edge enhancement
tvscl,roberts(img)
tvscl,sobel(img)
```



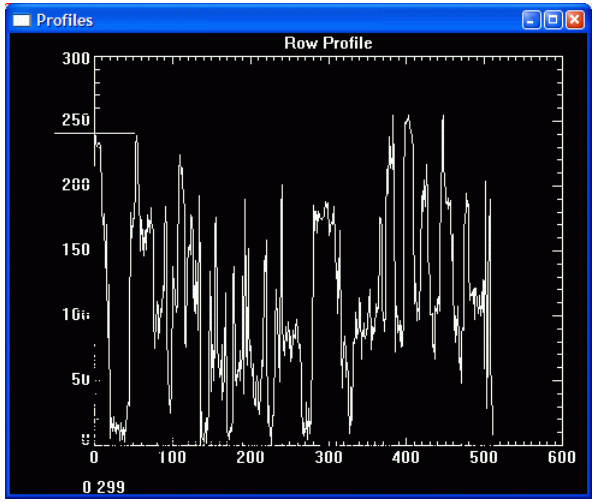


Fourier image processing

```
; experiment with Fourier filtering
img=read_image('Ikonos_pan.jpg')
s=size(img) & ncol=s(1) & nrow=s(2)
window,0,xsize=ncol,ysize=nrow & tvscl,img

window,1,xsize=ncol,ysize=nrow
freqimg=shift(fft(img,-1),ncol/2-1,nrow/2-1)
tvscl,alog(freqimg)

window,2,xsize=ncol,ysize=nrow
filter=make_array(ncol,nrow,/dcomplex) ; note that this zeroes initially
  dimx2=ncol/2L & dimy2=nrow/2L
  sigxy=make_array(ncol,nrow,/dcomplex,/nozero)
  x=(dindgen(ncol)-dimx2+1)
  dimy2m=dimy2-1L
  for i=0L,nrow-1L do begin
    y=replicate(double(i-dimy2m),ncol)
    sigxy(*,i)=x^2 + y^2
  endfor
umax=50d0
circ=where(sigxy lt umax^2)
filter(circ)=dcomplex(1,0)
filter=dcomplex(1,0)-filter  ; uncomment for high-pass
tvscl,filter

window,3,xsize=ncol,ysize=nrow
filtimg=freqimg*filter
tvscl,(abs(filtimg))

window,4,xsize=ncol,ysize=nrow
filtimg=fft(shift(filtimg,-(ncol/2-1),-(nrow/2-1)),1,/overwrite)
tvscl,filtimg
end
```

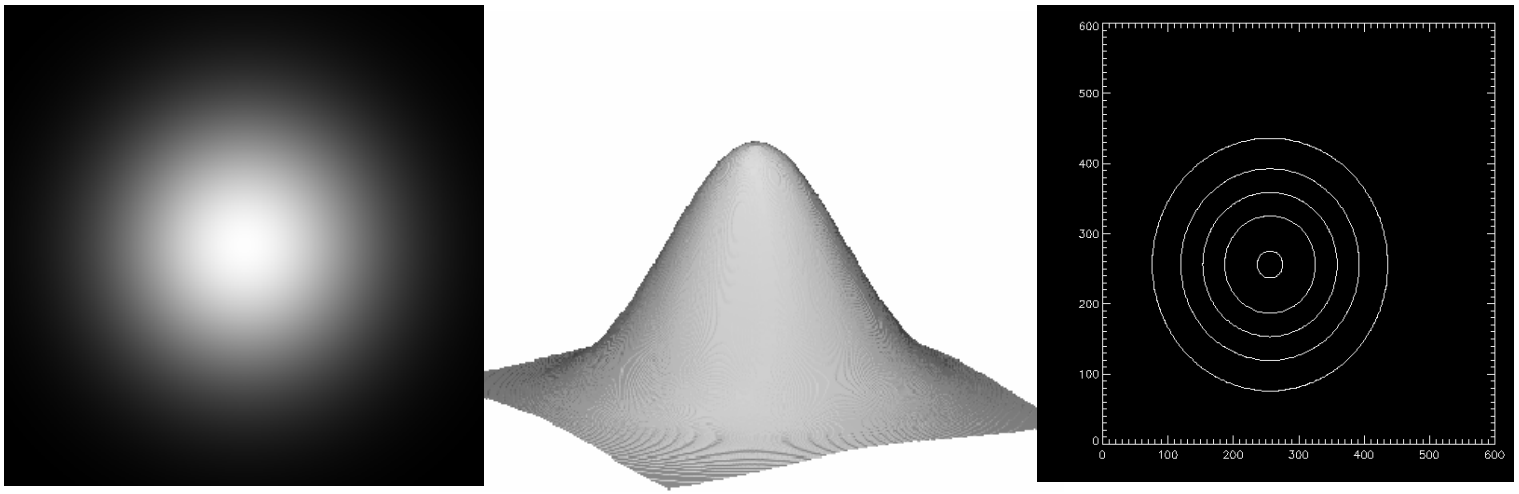
Image profiles

```
tvscl,img

; try…
profiles,img

; or…
p=profile(img)
plot,p
```



3d surfaces

```
window,xsize=512,ysize=512 ; create a window
x = findgen(512)-256
y = x
xy = exp(-x*x/20000.0) # exp(-y*y/20000.0)
img = byte(255.0*xy)
tv, img
shade_surf,img,xstyle=4,ystyle=4,zstyle=4
contour,img
```



Programming: functions, FOR loop

```
function factorial,k          ;program declaration
  f = 1
  for i=1,k do begin
        f = i * f
  endfor
  return, f
end
```

```
IDL> .run factorial
% Compiled module: FACTORIAL.
IDL> print,factorial(6)
       720
IDL> a=factorial(6)
IDL> print,a
       720
```

Function variables exist temporarily while the function runs. Variables explicitly passed back are accessible at the MAIN level.

Programming: procedures

```
pro factorial,k,f,ff,mult=mult          ;note use of keyword
  if (n_elements(mult) eq 0) then mult=1;if/then statement
  f = 1*mult
  for i=1,k do begin
        f = i * f
  endfor
  ff=f*f
end
```

```
IDL> .RUN factorial
% Compiled module: FACTORIAL.
IDL> factorial,6,a,a_squared,mult=2
IDL>  print,a,a_squared
      1440  2073600
```

Procedure variables exist temporarily while the procedure runs. Variables explicitly passed back are accessible at the MAIN level.

Programming: IF/THEN/ELSE

```
pro equality,a,b
  if (a EQ b) then begin
    print,'a,b',a,b
  endif else begin
    a=b
  endelse
  end
```

```
IDL> d=3 & e=7
IDL> equality,d,e
IDL> print,d
       7
```

EQ	equal
NE	not equal
LT	less than
LE	less than or equal to
GT	greater than
GE	greater than or equal to
NOT	not

Printing with IDL: postscript

- Printing with IDL

```
IDL> .run logistic
IDL> postscript,'log.ps'
IDL> .run logistic
IDL> postscript,/close
```

- This creates a printable file called **log.ps**
- Instead, can create file and automatically print it:

```
IDL> postscript,'log.ps'
IDL> .run logistic
IDL> postscript,/print
```

- Or view it (from command line): **% gs log.ps**
- Or print it (from command line): **% lpr log.ps**

Printing with IDL: images

- Printing with IDL

```
IDL> tvscl,img
IDL> buff=tvrd(true=3)
IDL> write_image,'myfig.gif','gif',buff
```

- This creates an image file called **myfig.gif**
- Include this in WORD or Powerpoint documents

- View with Netscape or other image viewing program such as **xv** or **display**, and then print using standard menu options
  - (from command line): **% xv myfig.gif**
  - (from command line): **% display myfig.gif**