# Contents



Text and programs available at `http://www.astro.uu.nl/~rutten/ssa`.

# Brief IDL manual

```
Why IDL?
--------
IDL is much used in astronomical image processing.
It is an interactive computer language with the following advantages:
 - programming language, not a package: make up your own stuff;
 - interactive: test out new tricks by trying them statement by statement;
 - array notation: a = a + (b < 0) works for whole multi-dimensional arrays.
The disadvantage is that IDL licenses are very expensive.  The Student
License is cheaper but limited to array size 65536 (256x256 pixel images).

Manuals
-------
The online help is good.
There are also complete IDL User's Guides in book form (3-4 big volumes).
The student version manual ("Learning IDL")  is compact but fairly good.
The IDL astronomy library sits at
   http://idlastro.gsfc.nasa.gov/homepage.html
Solar IDL software is found at:
   http://www.lmsal.com/ssw

IDL as interpreter
------------------
  IDL works directly on the command line when you hit return.
  This makes it easy to try new statements and statement sequences.
  The up cursor arrow brings back earlier commands.
  Try out the following IDL statements on the command line:

; on-line help
  ?                     ; inspect some IDL routines

; number games
  print,3*5             ; semicolon = comment (IDL skips rest of line)
  a=3*5
  help,a                ; show nature and value of this variable
  help,A                ; IDL is case-insensitive
  d=32767               ; integers run from -32768 to + 32768
  print,d+1             ; so what happens here?
  print,d+1.
  print,3/5
  print,3/5.            ; one floating number makes the result a float
  a=[1,2,3,4,5,6]       ; IDL variables can be (multi-dimensional) arrays
  a=[a,7]               ; lengthen array A by adding value at its end
  print, a, 1E6*a       ; single precision: 6 significant digits, <10^38
  print, a, 1D6*a       ; double precision: 16 significant digits
  print, a, 1/a
```

```
  print, a, 1./a
  print, a, a^2
  print, a, alog10(10^a)
  a=1.*a                 ; convert into floats
  print, a, alog10(10^a)
  print, a, alog(exp(a))
  print, a, acos(cos(a))
  print, a, a mod 2
  b=sqrt(a)             ; type of B is defined through its assignment
  help                  ; help without variable shows all variables
  print,'b=',b          ; 'something' is a string

; simple arrays
  a=intarr(100)         ; define A as integer array A(0),....,A(99)=0
  a=a+1                 ; now they are all A(i)=1
  for i=0,99 do a(i)=i  ; ==> BEWARE <==  IDL starts counting at 0!
  a=fltarr(100)         ; define A as floating number array A(0),...,A(99)
  a=dblarr(100)         ; double-precision float array
  a=indgen(100)         ; direct definition of A=0,1,....,99
  print,a(0),a(99)
  print,a(10:19)
  b=sqrt(float(a))
  for i=0,99 do if (b(i) GT 5) then a(i) = a(i) + b(i)
  a = a + (b gt 5)      ; same computation but much faster
  a = a + (b > 5)       ; same computation but yet faster

; simple plotting
  plot,sin(a/10)
  a=findgen(100)        ; float array A=0,1,....,99
  plot,sin(a/10)
  b=sin(a/5)/exp(a/50)  ; how many elements has B?
  plot,b
  plot,b,cos(a/5)
  oplot,b,cos(a/4)      ; over-plots in existing graph
  erase                 ; clean window
  x=b
  y=cos(a/5)
  plot,x,y,psym=1       ; defined for psym=1-7,10; try them out
                        ; something=something: optional "keyword" parameter
                        ; check PLOT in Online Help; check GRAPHICS KEYWORDS
  plot,x,y,linestyle=1  ; defined for linestyle=0-5
  plot,x,y,xtitle='x-axis',ytitle='y-axis',thick=2, $
      charsize=1.5            ; the $ extends command to next line
  plot,x,y,xrange=[-2,+2],yrange=[-1.5,1.5]   ; your choice of axes
  plot,shift(abs(fft(b,1)),25),/ylog
                        ; power spectrum on linear-log scales
                        ; /ylog is alternative for ylog=1 (true)
  xyouts,10,30,'Power Spectrum'           ; annotation, x,y in data units
```

```
  xyouts,.75,.75,'Power Spectrum',/normal  ; annotation, axis-length units

; more displays
  z=b#a                  ; z(i,j)=b(i)*a(j); full matrix multiplication: ##
  help,z                 ; so this is a 2-dim (100,100) float array
  print,z[0:5,0:5]       ; both square brackets and parentheses permitted
  plot,indgen(50),z[2,*]    ; plot 3rd column of array z
  plot,indgen(50),z(*,49)   ; plot bottom row of array z
  contour,z              ; just for fun
  surface,z              ; just for fun
  shade_surf,z           ; just for fun - but only in a small window
  window,1,xsize=200,ysize=200   ; open small window, size in pixels
  shade_surf,z
  wdelete,1              ; delete window 1
  show3,z                ; just for fun

; PostScript output
  set_plot,'ps'          ; now write to PostScript file instead of window
  device,filename='surface_1.ps'    ; choose filename
  show3,z                ; all IDL output commands now write to that file
  device,/close          ; done
  set_plot,'win'         ; back to output on Windows95 screen
  set_plot,'x'           ; idem for Unix
  help,/device           ; ,/device is the same as using ,device=1 (enable)

; image display
  tv,z                   ; show array as image
  zsize=SIZE(z)          ; get array type and size
  nx=2*zsize(1)
  ny=2*zsize(2)          ; etcetera for more dimensions; zsize(0)=type
  z=rebin(z,nx,ny)       ; resample z to make its display bigger
  wdelete
  window,xsize=nx,ysize=ny      ; window equal to image size
  z8=bytscl(z)           ; scales dynamic range to 8 bits (shades 0 - 255)
  print,min(z),max(z),min(z8),max(z8)    ; list extrema
  tv,z8                  ; display image with maximum contrast on screen
  tvscl,z                ; same as tv,bytescl(z)
  profiles,z             ; measurent tool for showing image values
                         ; stop it with right mouse button (cursor on image)
  adjct                  ; tool to adjust greyscale table
  xpalette               ; tool to adjust color table
  xloadct                ; idem
  tvscl,z8>127           ; display only the brighter half of the pixels
  erase
  tvscl,z(0:nx/2,0:ny/2)     ; bottom-left quarter of the image
  wdelete
  tvscl,congrid(z,188,188,/interp) ; arbitrary resizing with interpolation
```

```
; read/write files
  openw,1,'myfile.ext'  ; open file myfile.ext on logical unit 1 for writing
  printf,1,z            ; write free-format file
  close,1               ; free lun 1
  openr,1,'myfile.ext'  ; now open that file for reading
  zz=z                  ; define variable type and size
  readf,1,zz            ; read free-format file
  close,1
  (use: - WRITEU, READU to write/read large files unformatted = faster;
        - image=assoc(lun,bytarr(nx,ny)) to get single images from a movie)

; saving IDL command sequences
  journal,'filename'     ; copies all typed commands to a file
  @filename              ; executes a sequence of IDL commands in a file
  save,'filename.sav'    ; saves a full session (not in Student Version)
  restore,'filename.sav' ; restores a session save set

Procedures and functions
------------------------
  Start a new file pract.idl or procedurename.pro or functionname.pro;
  edit it (Windows: IDL desktop; Unix: external editor or idlde).

  PRO procedurename, param1, param2
    ; this one does this...
    ; parameters: param1 = etc   ; input value or input/output parameter
    IDL statements
    IDL statements
  END

  FUNCTION functionname, param1, param2, param3, ....
    ; this one does this...
    ; parameters: param1 =etc    ; input value or input/output parameter
    IDL statements
    IDL statements
    something=...
    RETURN,something
  END

  IDL statement      ; start of main-level program
  IDL statement
  STOP               ; continue after this STOP with .CON

  IDL statement
  IDL statement
  END

  A sequence of IDL statements after the last procedure or function
  that does not start with PRO or FUNCTION makes up the main level
```

```
  program.  It should end with END.  You run it with .R FILENAME.

  After program completion all variables are available for inspection
  on the command line.  Use this main level for trying out new things.
  When such a sequence is complete, then turn it into a procedure or
  function.  The latter may go to a separate NAME.PRO file or may remain
  in your main FILENAME.IDL file, above the statement part.

Using procedures and functions
------------------------------
  IDL> .run procedurename          ; compilation (only main program is run)
  IDL> procedurename,param1,...    ; run compiled procedure

  IDL> .com functionname.pro       ; compilation
  IDL> a=functionname(param1,...)  ; evaluate compiled function

  example (in a separate file ADDUP.PRO):
    function addup,arr
      ; sums linear array ARR - but IDL TOTAL is faster and more general!
      arraysize=SIZE(arr)
      if (arraysize(0) ne 1) then print,'addup input is not a 1D array'
      sumarr=0
      for i=0,arraysize(1)-1 do sumarr=sumarr+arr(i)
      return,sumarr
    end
    IDL> .run addup               ; recompile after every program change
    IDL> try=findgen(100)         ; floats 0.,......,99.
    IDL> print,addup(try)
    IDL> print,total(try)         ; check with IDL array summation

  "Disappearing variables": after an error in a procedure or function your
  session stops within that procedure/function.  HELP displays only the
  local variables.  RETURN gets you one level higher; RETALL gets you
  back to the top level.

  STOP in a procedure/function/main stops it there to let you inspect the
  local variables at that place in the statement sequence.
  Continue with  .continue (or .con).

Program structures
------------------
    if i>16 then begin
      IDL statement
      IDL statement
    endif else begin
      IDL statement
      IDL statement
    endelse
```

```
if ( y eq 3) then x=2 else x=1  ; relational operators: EQ NE LE LT GE GT

for j=0,9 do number(j)=sin(region[j] * !pi)    ; ! gets system variable

for j=0,20,2 do begin
  number[j] = sin(region[j] * !pi)
  region[j]=0
endfor

while (a and (cnt ne 0)) do begin        ; logical operators: AND OR XOR
  print, 'Still going at count: ', cnt
  cnt=cnt-1
endwhile

if n eq 0 goto, finish
IDL statement
IDL statement
finish:
return
  ; but since good programmers don't use goto's, a better solution is:
  if n neq 0 then begin
    IDL statement
    IDL statement
  endif
  return
```

```
Programming style hints
-----------------------
  - don't forget that IDL array indices start at 0
  - split programs in separate procedures and functions
  - use parameters instead of numbers to get dynamical adaptivity
  - use SIZE(array) to get array dimensions for general procedures
  - choose clear variable names
  - add lots of comments (in English please)
  - add detailed explanation at procedure/subroutine start
    (some libraries do this between ;+ and ;- lines and cut out these
     comment blocks for automatic manual generation)
  - answer a routine call without parameters or () with a specification:
    IF N_PARAMS(0) LT 1 THEN BEGIN
      PRINT,'procedure xxxx requires parameters: yyy, zzz'
      RETURN
    ENDIF
  - indent BEGIN ... END structures
  - use capitals for IDL terms, routines and operators
  - use capitals for image arrays in image processing
  - JOURNAL,'filename' records all your command-line entries, useful for
    subsequent conversion into programs.
```