Introduction to IDL

Wolfgang Dobler

Wolfgang.Dobler@ncl.ac.uk

http://antares.ncl.ac.uk/~dobler/doc/idl/

November 22, 2001

Contents

Foreword

IDL

- $\approx$ least elegant programming languages I have used
- (but I never used *COBOL* or *Visual Basic* . . .)
- most powerful graphics tool I have ever used:
  - ○ full-featured programming language
  - $\leadsto$ some aspects are awkward, but you can program around them
- widely used in the scientific community
- interactive
- optional arguments for functions and procedures

Similar tools

Matlab
Octave (a Matlab clone)
- much more modern language
- much lower graphics quality

SciLab (another Matlab clone)
PerlIDL (Perl derivate)
- very powerful language (Perl)
- still quite beta

Python-numeric / Python-scientific (Python library)
Yorick

⋮

Xmgr/Grace
- click-and-cramp
- apparently has a scripting language

Gnuplot
- scripting language, but not more

⋮

Help

Command line: *idlhelp* $\Longrightarrow$ online help

In IDL: Type *'?'* $\Longrightarrow$ online help

In IDL: '`idl> help, var`' $\Longrightarrow$ info on *`var`*

Literature:

- David Fanning, *IDL Programming Techniques*, 2nd Edition, 2000; ISBN 0-9662383-2-X

Web:

- `http://www.dfanning.com/` (*very* useful)
- RSI Technical Tips at `http://www.rsinc.com/services/prodspec.cfm?product=IDL`

  - (Very old) FAQ list at `http://afm1.lbl.gov/idl_faq.html`
  - Newsgroup `news:comp.lang.idl-pvwave`

Online handbooks: PDF manuals come with the IDL installation, starting point: *$IDL/docs/onlguide.pdf*

---

1. Data Types

Atomic:

- byte, integer, long (integer)
- float, double, complex
- string

Complex:

- arrays
- structures
- object classes

Beware of

```
for i=0,100000 do
```
(something)

— the 2-byte integer `i` will never attain the value 100000. Instead write

```
for i=0L,100000 do
```
(something)

Now `i` is initialised as *long int* and your loop will (eventually) finish.

Remarks

- not declarative
- 2/3 problem (shared with C, Fortran, . . .):
  $2./3. \neq 2/3 = 0$
- Examples:
  - `x = 5` *(integer)*
  - `x = 5D0` *(double)*

    - `z = complex(x,7)` *(guess what)*
- info on variables:
    - `help, x`
    - `help, !p, /STRUCT`
- system variables: ‘!p’, ‘!x’, ‘!y’, ‘!z’, ‘!d’, ‘!pi’ (and others)

Arrays

- `zeros = fltarr(10,20)`
- `ones = make_array(10,20,VAL=1)`
- zero indexing: `zeros[0,0] = 1`
- `count = indgen(10) & print, count`

- coordinate vectors

```
nx=50 & ny=60 & nz=70

x = findgen(nx)   ;; x = 0, 1, . . . , 48, 49

x0=-1. & x1=1.
x = x0 + findgen(nx)/(nx-1.)*(x1-x0)
   ;; x = -1, -0.86, . . . , 1

   ;; alternatively:
x = linspace(-1, 1, nx)   ;; (my routine)
y = linspace(-1.5, 1.5, ny)
z = linspace(0.2, 5, nz)
```

- *rebin-reform*
  reform: re-shape array without changing data
  rebin: duplicate array elements

```
xx = rebin(reform(x, nx, 1), nx, ny)
```
;; coordinate grid array
```
yy = rebin(reform(y, 1, ny), nx, ny)
rr = sqrt(xx^2+yy^2)
```

– array syntax

much faster than explicit looping

```
ff = sin(6*xx)*exp(-2*rr)
surface, ff, x, y
```

– array slices:

- `ff[0,0]`, `ff[10,7]`
- `ff[2:5,0]`
- `ff[*,5]`
- `ff[*,3:7]`

– *where* function and array subscripts:

```
bad = where(rr gt 0.5)
```
;; don't use '>' instead of 'gt' $\Longrightarrow$ *dubious results*
```
ff[bad] = 0
surface, ff, x, y
```

---

2. Plotting

1-dimensional

```
f = sin(3*x)*exp(-x)
plot, x, f
plot, x, f, XRANGE=[0,1], COLOR=150
```

```
plot, x, f, PSYM=-4
plot, x, f, PSYM=10

g = sqrt(2*!pi*z)*z^z*exp(-z)
plot, z, g
plot, z, g, /XLOG, /YLOG
plot, z, g, /YLOG
oplot, z, gamma(z+1), LINESTYLE=2, COLOR=150
xyouts, 0.5, 1.5, "Stirling’s formula"
```

2-dimensional

```
surface, ff, x, y
for i=0,360,10 do begin $
   surface, ff, x, y, AZ=25+i & wait, 0.1

shade_surf, ff, x, y

xsurface, ff

contour, ff, x, y
contour, ff, x, y, /FILL
contour, ff, x, y, /FILL, NLEVELS=60

;; (More or less) the same, but shorter:
contourfill, ff, x, y, /GRID   ;; (my routine)
```

```
gg = cos(xx)*exp(-rr)
velovect, ff, gg, x, y
vel, ff, gg
vel, ff, gg, LEN=0.2, NVECS=1000
```

Combining different types of plotting:

```
  contourfill, ff, x, y
  contour, ff, x, y, NLEVELS=20, /OVERPLOT
  velovect, ff, gg, x, y, /OVERPLOT
```

3-dimensional

```
  xxx = rebin(reform(x, nx, 1, 1), nx, ny, nz)
  yyy = rebin(reform(y, 1, ny, 1), nx, ny, nz)
  zzz = rebin(reform(z, 1, 1, nz), nx, ny, nz)
  rrr = sqrt(xxx^2+yyy^2)
  phi = atan(yyy,xxx)
  m = 1
  kz = 2*!pi/(z[nz-1]-z[0])
  fff = rrr^2*exp(-4*rrr^2)*cos(m*phi-kz*zzz)

  shade_volume, fff, 0.9*max(fff), vert, poly
  scale3, $    ;; ($-sign = continuation character)
    XRANGE=[0,nx], YRANGE=[0,ny], ZRANGE=[0,nz]
  image = POLYSHADE(vert, poly, /T3D)
  loadct, 3
  TV, image
```

Key words vs. environment variables

```
  f = cos(z)
  plot, z, f, XRANGE=[0,6]
```

vs.

```
  !x.range = [0,6]
  plot, z, f
  plot, z, sin(z)
```

!x is a *structure* and *!x.range* accesses one slot of it:

```
help, /STRUCTURE, !x
```

keyword	*env. variable*
`title`	`!p.title`
`color`	`!p.color`
`charsize`	`!p.charsize`
`linestyle`	`!p.linestyle`
`psym`	`!p.psym`
`thick`	`!p.thick`
	`!p.multi`
`{x,y,z}charsize`	`!{x,y,z}.charsize`
`{x,y,z}margin`	`!{x,y,z}.margin`
`{x,y,z}range`	`!{x,y,z}.range`
`{x,y,z}style`	`!{x,y,z}.style`
`{x,y,z}title`	`!{x,y,z}.title`

Colour tables

```
contourfill, ff, x, y
loadct, 5   ;; loads colour table No. 5
contourfill, ff, x, y
loadct, 16   ;; loads colour table No. 16
contourfill, ff, x, y
xloadct   ;; interactively pick colour table
```

Colour problems

If you only get different shades of red, try

```
device, DECOMPOSE=0
```

in your IDL startup file (☞ below).

Windows and frames

Open a new window:

```
  window, 1
```

Plot several graphs in one window

```
  !p.multi = [0,3,2]
  for i=0,5 do plot, x, x^i, XRANGE=[0,1]
  !p.multi = 0    ;; reset to single plot
```

Hardcopies

```
  set_plot, ’PS’
  plot, z, f
  device, /CLOSE
  set_plot, ’X’
  ;; or (my commands):
  psa, FILE=’tmp.ps’, THICK=2
  plot, z, f
  pse
```

Fonts

```
  plot, x, f, XTITLE=’!8B!6!Dnorm!N - !7w’
```

You can also use PostScript fonts (requires some setup; default with my *psa*, *pse*) or TrueType fonts

3. Files and Functions

Files

Write

```
x1 = linspace(0,10,50)
y1 = cos(x1)
```

to file *incl1.pro* and

```
@incl1
plot, x1, y1
end
```

to file *short.pro*

Now you can run it with

```
idl>  .r short
```

You can however *not* run *incl1.pro* this way:

```
idl>  .r incl1
% End of file encountered before end of program.
```

since the end is missing $\Longrightarrow$ inconsistency.

Functions and subroutines

Write

```
function htan, x
  if (x lt 0) then begin
    res = tanh(x)
  endif else begin
    res = tan(x)
  endelse
  return, res
end
```

to file *htan.pro* and

```
pro jabber, x, y, z, BRILLIG=bril
  if (keyword_set(bril)) then print, ’Brillig’
  print, ’(x,y,z) =’, x, y, z
end
```

to file *jabber.pro*

Now you can use the new function *htan*

```
idl>  print, htan(0.7)
```

and procedure *jabber*

```
idl>  jabber, 5, 3, htan(-2)
```

Simple real-life examples

(yet simplified)

idl/lib/default.pro

```
pro default, var, val
  if (n_elements(var) eq 0) then var=val
end
```

idl/lib/minmax.pro

```
function minmax, f
  on_error, 2    ;; return to caller on error
  return, [min(f),max(f)]
end
```

idl/lib/contourfill.pro

```
pro contourfill, z, x, y, $
               NLEVELS=nlevels, _EXTRA=_extra
  if (n_elements(nlevels) eq 0) then nlevels=60
  contour, array, x, y, $
    NLEVELS=nlevels, /FILL, _EXTRA=_extra
end
```

Startup file; journalling

You want to be able to use your own procedures from everywhere.

1. Put your (general purpose) scripts into directory `~/idl/lib/` (or `~/idl/pro/`, `...`)
2. Tell *IDL* to read *~/.idlrc* at startup (in *~/.cshrc*):

   ```
   setenv IDL_STARTUP $HOME/.idlrc
   ```

3. Add your directory to the *IDL* search path. In *~/.idlrc*, write

   ```
   device,decompose=0  ;; (probably needed for indexed colour)

   !EDIT_INPUT = 1000  ;; (increase length of history)

   !path = !path + $
     ':~/idl/lib:' + expand_path('+~dobler/idl/lib') + $
     ':~brandenb/idl/pro'
     ;; (append your and others' directories to search path)
   ```

Journalling creates a script of your IDL session

⟹ turn experiments into scripts by adding ' end '

```
idl>  journal, 'jou.pro'  ;; activate journalling

;; (interactively try some IDL statements)

flush, !journal  ;; ensure journal file is up-to-date

;; (copy journal file, edit if necessary and add ' end ')
;; (. . . )

idl>  journal  ;; deactivate journalling
```

Subroutines vs. working in global scope

Subroutines:

- `sub1, a, b, c`
- allow for good programming style
- local variables ($\Longrightarrow$ no name clashes)
- need *common blocks* for global communication

Global scope:

- `.r glob1`
- interactive access to *all* data (≙ idea of an interactive language)
- caution needed: don't overwrite variables

Recommendation: use subroutines for general-purpose tasks only; work in global scope with your data

4. Interacting with Fortran

```
Fortran code                          IDL program
  real, dimension(5,7,7) :: a           a = fltarr(5,7,7)
  double precision :: d                 d = 0D0
  integer, i,k,l                        i=0L & j=0L & k=0L
    ...                                   ;; Fortran integers are
                                          ;;   IDL long ints

                                        close, 1   ;; (just to be sure)
                                        openu, 1, /F77
                                          ;; open unformatted
                                          ;; assume F77 records
  write(1) a, i                         read, 1, a, i
  write(1) d, k, l                      read, 1, d, k, l
                                        close, 1
```

Doing it all in *IDL*

Philosophy: Want to be able to do the same things with your data as in Fortran

Thus (to work with finite-difference code): need derivative operators (*xder*, *xder2*, etc.)

Add time-stepping $\Longrightarrow$ don't need Fortran at all.

Example: advection of passive scalar

File *start.pro*:

```
;; start.pro  ---  Initialisation

COMMON cdat, x,y,z,nx,ny,nz,nw,ntmax,date0,time0
COMMON params, visc,u0

@xder_6th_1d  ;; load appropriate derivative routines
@xder2_6th_1d
@pde          ;; compile equations
@rk           ;; simple Runge-Kutta scheme

;; Parameters
nx = 50
u0 = 1
cs2 = 1.
;; Grid
x = linspace(0,1,nx,/PERIODIC)
dx = x[1]-x[0]
dt = 0.4*dx/u0     ;; time step
visc = 0.005*dx*u0 ;; numerical viscosity
;; Initial condition
f = tanh(5*cos(2*!pi*x)) & t = 0

end
```

File *pde.pro*:

```
;; pde.pro  ---  Equation(s) for advection
function pde, f
  COMMON cdat, x,y,z,nx,ny,nz,nw,ntmax,date0,time0
  COMMON params, visc,u0
;
  dfdt = -u0*xder(f) + visc*xder2(f)
  return, dfdt
end
```

File *run.pro*:

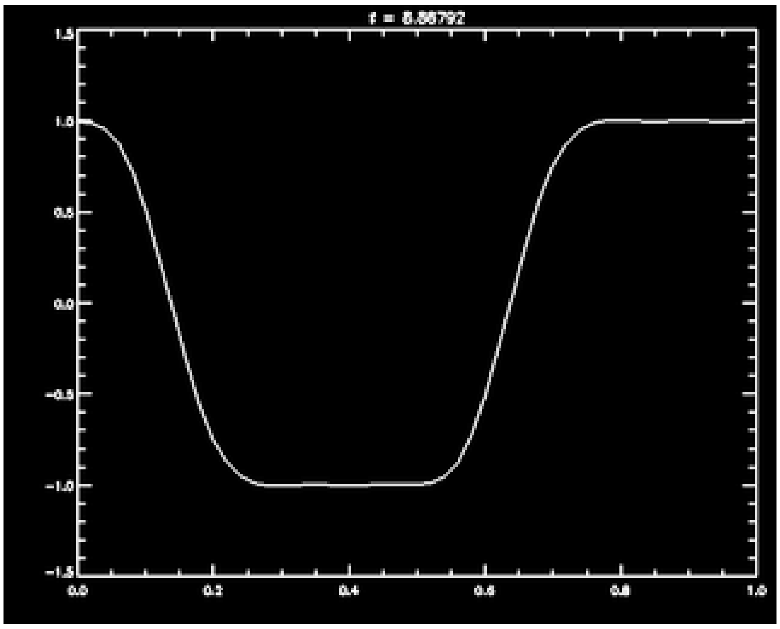
```
;; run.pro  ---  Time-stepping and plotting

for i1=0,100 do begin
  for i2 = 0,10 do begin
    rk, f,t,dt
  endfor
  plot, x, f, TITLE='!8t !3= '+strtrim(t,2)+'!X'
  wait, .1
endfor

end
```

Now run this code:

```
idl>  .r start
idl>  .r run
idl>  .r run
```


t = 8.88792
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0