Tutorial de IDL básico

Salvador Ramirez Flandes
sram@profc.udec.cl

28 de junio de 2001

Índice General

Índice de Tablas

Capítulo 1

Introducción

IDL (Interactive Data Language) es un lenguaje interpretado, orientado al arrelo, de análisis matemático y de despliegue gráfico.

Interpretado significa que existe un programa intérprete que procesa el código escrito en este lenguaje. Por esta razón no es posible directamente generar un programa ejecutable a partir de un programa hecho en IDL, y los programas entonces dependen de la existencia del intérprete en la plataforma sobre la cual se deseen correr los programas[1].. Un programa realizado en un lenguaje interpretado tiene sus ventajas y desventajas. Una de las principales ventajas es la portabilidad sobre plataformas distintas, es decir, un programa, escrito en IDL por ejemplo, creado en Unix debería correr sin problemas en Windows si se dispone del intérprete. Entre las desventajas más importantes están que los programas son mas lentos (comparados con programas ejecutables) y que no es posible proteger el código (cualquier usuario que tenga el programa puede ver como está programado).

Un lenguaje orientado al arreglo es permite que los operadores puedan interactuar directamente sobre arreglos sin tener que programar un loop para recorrer sus elementos.

IDL además posee numerosas rutinas de análisis numérico y estadístico que evita al programador tener que estarlas programando o usando librerias externas. IDL también soporta varios formatos tanto de imágenes (bmp, gif, jpeg) como de almacenamiento de datos científicos (cdf, hdf, netcdf[2]).

Este manual asume que el lector posee conceptos básicos de programación.

[1] Actualmente IDL posee un libreria runtime para muchas plataformas, que permite correr programas IDL sin necesidad de tener instalado el programa IDL completo.

[2] CDF significa Common Data Format y HDF significa Hierarchical Data Format.

Capítulo 2

Estructura del Lenguaje

2.1 Tipos de datos

Un tipo de dato sin signo significa que usa todos los bits que posee en direccionar sólo números positivos. Por ejemplo Byte, que posee 8 bits, puede direccionar de 0 a 2^8 (255). El Integer en cambio, puede direccionar desde $-\frac{2^{16}}{2}$ a $(\frac{2^{16}}{2}-1)$. Este último posee un rango no simétrico por direccionar el cero.

2.2 Arreglos

Los arreglos más usados son vectores y matrices. Los vectores son colecciones unidimensionales de datos, mientras que las matrices son colecciones bidimensionales.

En IDL las matrices (y los arreglos en general, aunque en arreglos de más de 3 dimensiones los conceptos de columnas y fila pierden el sentido natural) son de la forma A[*columna,fila*]. Donde *columna* y *fila* comienzan desde cero.

$$\begin{pmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{pmatrix}$$

Tipo de datos	Definición
Byte	Entero de 8 bits (sin signo)
Integer	Entero de 16 bits (con signo)
Longword	Entero de 32 bits (con signo)
Floating Point	Número real de 32 bits de simple precisión
Double Precicion	Número real de 64 bits de doble precisión
Complex	Número complejo de simple precisión
Double Precision Complex	Número complejo de doble precisión
String	Secuencia de caracteres

Tabla 2.1: Tipos de datos en IDL

Tipo de dato	Crear arreglo vacio	Crear arreglo de indices
Byte	B = bytarr(d1,...,dn)	B = bindgen(d1,...,dn)
Integer	I = intarr(d1,...,dn)	I = indgen(d1,...,dn)
Longword	L = lonarr(d1,...,dn)	L = lindgen(d1,...,dn)
Float	F = fltarr(d1,...,dn)	F = findgen(d1,...,dn)
Double	D = dblarr(d1,...,dn)	D = dindgen(d1,...,dn)
Complex	C = complexarr(d1,...,dn)	C = cindgen(d1,...,dn)
Double complex	DC = dcomplexarr(d1,...,dn)	DC = dcindgen(d1,...,dn)
String	S = strarr(d1,...,dn)	S = sindgen(d1,...,dn)

Tabla 2.2: Funciones para la creación de arreglos

A continuación se presentan algunos ejemplos de creación de arreglos directamente de los datos que los componen:

Arreglo unidimensional: `A = [ 1.0, 4.5, 6.7, 4.9 ]`

Arreglo bidimensional: `A = [ [1,2,3], [4,5,6], [7,8,9] ]`

Para crear arreglos mas grandes se pueden usar las funciones de la tabla 2.2. En estas funciones, los parámetros pasados son las dimensiones del arreglo deseado. La cantidad máxima de dimensiones de un arreglo en IDL es 8, por lo que $n \leq 8$ en todas esas funciones. En estas mismas funciones, se entiende por arreglo vacio a un arreglo con valores por defecto para el tipo de dato respectivo. Por ejemplo, en los float el valor por defecto es 0.0 mientras que en los strings es el string nulo. Los índices de los arreglos de índices son el número correspondiente dentro del string.

Tal como se puede apreciar en el último ejemplo, la operación de división por 2 se aplicó directamente y de una sóla vez a todos los elementos del arreglo que entregó FINDGEN, esto es lo que se mencionaba en la introducción de que IDL es orientado al arreglo al evitar al programador tener que crear un loop para recorrer el arreglo para aplicar alguna operación a sus elementos. De esta misma forma, otras operaciones matemáticas también son posibles de aplicar a todos los elementos de un arreglo:

```
A = FINDGEN(2,3)
X = SQRT(A/3)
```

Si se quiere solo una parte del arreglo entonces se puede hacer:

```
X = SQRT( A[0:1, 0:2]/3 )
```

A[0:1, 0:2] significa el sub-arreglo comprendido entre la columna 0 y 1 y entre la fila 0 y 2.

Para averiguar el tamaño o las dimensiones de un arreglo existen las funciones N_ELEMENTS y SIZE. Ejemplos:

1. ```
   X = FINDGEN(3,3)
   PRINT, N_ELEMENTS (X)
   → 9
   ```

2. ```
   X = BINDGEN(4,6,7)
   PRINT, SIZE (X)
   →  3  4  6  7  1
   ```

El comando SIZE en realidad entrega un vector de valores que se especifican en la tabla 2.3. Los códigos de los tipos de datos se muestran en la tabla 2.4.

Elemento vector S	Significado
S[0]	Número de dimensiones
S[1]	Valor primera dimensión
S[2]	Valor segunda dimensión
S[3]	Valor tercera dimensión
S[4]	Código del tipo de dato de los valores del arreglo

Tabla 2.3: Salida de la función SIZE (S = SIZE(X))

Código	Tipo de dato asociado
0	no definido
1	byte
2	integer
3	longword integer
4	floating point
5	double precision float
6	complex floating
7	string
8	structure
9	double precision complex float
10	pointer
11	object reference

Tabla 2.4: Códigos de tipos de datos devueltos por SIZE

2.3 Procedimientos y Funciones

Los procedimientos y funciones son pedazos de código independientes para hacer una tarea determinada. La diferencia entre funciones y procedimientos en IDL es que estos últimos no retornan un valor, mientras que las funciones si. Los procedimientos tienen la siguiente forma:

```
PRO NombreProcedimiento
Cuerpo del Procedimiento
END
```

Las funciones se declarar asi:

```
FUNCTION NombreFuncion
Cuerpo de la Funcion
RETURN, ValorARetornar
END
```

El procedimiento principal del programa (del cual se llaman los demás procedimientos y funciones) debe llamarse con el nombre del archivo que lo contiene. Por ejemplo si existe un archivo IDL llamado *programaIDL.pro* entonces dentro de este archivo debe existir un procedimiento principal llamado *programaIDL*, es decir, debe haber una declaración así: *PRO programaIDL*.

Los parámetros pasados a un procedimiento deben ir separados por comas y sin paréntesis. Ejemplo: PRINT es un procedimiento en IDL:

```
PRINT, ´Esta es´, ´una´, ´prueba:´, X
```

Los parámetros pasados a una función en IDL deben ir entre paréntesis y separados por comas también. Ejemplo: FLTARR es una función en IDL:

```
X = FLTARR(3,2)
```

2.4 Ciclos de control e iteración

Los más usados en IDL son el *for* y el *if*. Su sintaxis es la siguiente:

```
FOR Variable=Expresión1, Expresión2 DO BEGIN
 Código IDL
ENDFOR

IF Condición THEN BEGIN
 Código IDL
ENDIF

IF Condición THEN BEGIN
 Código1 IDL
ENDIF ELSE BEGIN
 Código2 IDL
ENDELSE
```

Cuando el código dentro de FOR o IF se pueden expresar en una sóla línea entonces no es necesario BEGIN ni ENDFOR ni ENDIF. Por ejemplo:

```
FOR i=0, N-1 DO $
```

Nombre expresión	Descripción
eq	equal
ne	not equal
le	less than or equal
lt	less than
ge	greater than or equal
gt	greater than

Tabla 2.5: Lista de operadores de comparación

Constantes IDL	Valor	Descripcion
!PI	π	π con precision simple
!DPI	π	π con doble precision
!DTOR	$\pi/180$	factor para pasar grados a radianes
!RADEG	$180/\pi$	factor para pasar radianes a grados

Tabla 2.6: Constantes en IDL

```
   IF (A[i] le 2.5 ) THEN A[i-1] = A[i] * i

FOR i=0, N-1 DO $
   IF (A[i] le 2.5 ) THEN $
     A[i-1] = A[i] * i
```

Las expresiones de comparación se listan en la tabla .

2.5 Constantes útiles en IDL

Las constantes matemáticas de IDL se muestran en la tabla 2.6.

Notar que la constante e no se encuentra definida pero puede ser fácilmente obtenida con: `e = exp(1)`.

Ejemplo uso de las constantes:

```
PRINT, acos(A) * !RADEG
```

Aparte de las constantes matemáticas numéricas presentadas en la tabla 2.6, existen otras constantes menos usadas pero muchas veces útiles también. Por ejemplo la estructura !VERSION posee información acerca de la plataforma sobre la cual se está corriendo un programa IDL. A continuación una breve descripción de algunos campos de esta estructura:

!VERSION.OS_FAMILY: que puede tener uno de los valores: *unix*, *windows* o *mac*.

!VERSION.OS: En el caso de Unix esta variable posee información mas específica: *linux, solaris, osf, irix, aix,* etc.

!VERSION.ARCH: información acerca de la arquitectura de la CPU: *i386, sparc*, etc.

!VERSION.RELEASE: información de la versión de IDL que se está corriendo: 5.2, 5.3, etc.

Capítulo 3

Graficando

3.1 Granficando curvas de dos dimensiones (plotting)

El procedimiento IDL para graficar curvas de dos dimensiones es *plot*, y su sintaxis es:

PLOT, [X,] Y [, *resto de opciones gráficas*]

Notar que el vector X es opcional (variable independiente del gráfico) y si no se especifica entonces Y se grafica como una función de sus subíndices en el orden de los elementos de Y. Por ejemplo:

Y = [4.5, 8.9, 2.1, 6.7]
Entonces Y es graficado dándole a X los valores:
X = [0.0, 1.0, 2.0, 3.0]

Si X se especifica entonces se gráfica simplemente Y en función de X, aunque no siempre el resultado es el gráfico de una función, dependiendo de los vectores X e Y que pueden representar una relación en lugar de una función. Esto se grafica así: `plot, X, Y`.

Para desplegar gráficos superpuestos se usa el procedimiento *oplot* (overplot). De esta manera el último gráfico queda dibujado encima del anterior (si es que había uno antes). Para desplegar varios gráficos en una misma ventana se debe modificar la estructura !P (plotting) de la siguiente manera:

!P.MULTI = [0, columnas, filas][1]

Por ejemplo, para graficar X1 y X2 :

```
!P.MULTI = [0, 2, 1]
PLOT, X1
PLOT, X2
```

[1] !P.MULTI es un arreglo que posee más elementos que manejan otro tipo de cosas, como por ejemplo, el lugar donde el primer plot será puesto. Sin embargo aquí sólo se verán los elementos más usados. Más información acerca de todas las opciones de esta variable se pueden encontrar en el manual de IDL o en la ayuda en línea.

Nombre variable	Descripcion
!P.BACKGROUND	Color de fondo (def:0)
!P.COLOR	Color de las lineas
!P.FONT	Fuente de letra
!P.LINESTYLE	Estilo de las lineas
!P.NOERASE	No borra el grafico anterior
!P.SUBTITLE	Subtitulo bajo el eje X
!P.THICK	Ancho de las lineas
!P.TITLE	Titulo principal del grafico

Tabla 3.1: Opciones de plotting

para volver a la normalidad: !P.MULTI = 0

La estructura !P también posee otros campos que manejan el comportamiento del *plotting*. La mayoría de esta opciones son también configurables por las opciones gráficas del procedimiento PLOT. Sin embargo al modificar las variables !P se obtiene un comportamiento de cambio permanente en tales opciones. En la tabla se muestran algunas variables de la estructura !P.

3.2 Graficando superficies tridimensionales

Los procedimientos para graficar superficies tridimensionales son: *surface y shade_surf*.

SURFACE, Z [,X, Y][,*resto de opciones de graficos*]

SHADE_SURF, Z [,X, Y][,*resto de opciones de graficos*]

SHOW3, Z [,X, Y][,,*resto de opciones de graficos*]

Ejemplos:

```
Z = SHIFT (DIST(40), 20, 20 )
Z = EXP (-(Z/10)^2 )
SURFACE, Z
SHADE_SURF, Z
```

Ahora girando un poco los ángulos de rotación de los ejes X & Z, poniendo título al gráfico y graficando sin borrar el gráfico anterior:

```
SURFACE, Z, ax=70, az=25, title=´Título de prueba´, /NOERASE
```

También existe una opción para que SHADE_SURF coloree la superficie de acuerdo a la elevación de ella:

```
SHADE_SURF, Z, shade=BYTSCL(Z)
```

Ahora para cambiar los colores usados para el coloreo se usa el procedimiento LOADCT de la siguiente manera:

```
LOADCT, ColorTable
```

donde *ColorTable* es un número que empieza en 1 y va hasta la cantidad de tablas de colores que hayan, así:

```
LOADCT, 20
SHADE_SURF, Z, shade=BYTSCL(Z)
```

Si lo anterior no hace cambiar de colo al gráfico es porque IDL por defecto usa tablas de colores de 8 bits, así puede ser que se alcancen a ver los demás colores de la tonalidad. Para cambiar eso se puede ingresar el siguiente comando:

```
DEVICE, decompose=0
```

Capítulo 4

Entrada y Salida

La E/S es uno de los temas más importantes de un lenguaje de programación para un científico, pues normalmente él obtiene los datos de medidas de instrumentos.

4.1 Salida a pantalla y entrada desde teclado

Para imprimir una expresión (un dato, un arreglo o una operación sobre alguno de los anteriores) en pantalla simplemente usamos el procedimiento IDL PRINT, de la siguiente forma:

PRINT, *expresión1 [,expresión2, ... , expresiónN]*

Por ejemplo para imprimir los elementos del 5 al 20 y el 30 de un vector X y al final un mensaje de texto como ´hola mundo´, entonces hacemos:

```
PRINT, X[4:19], X[30], ´Hola mundo´
```

Para leer un dato desde teclado usamos READ:

```
READ, variable, PROMPT=´Mensaje:  ´
```

Por ejemplo, para pedir al usuario que ingrese un número:

```
s = 0
READ, s, PROMPT=´Ingrese un número:  ´
```

4.2 Entrada y salida a archivos en formato ASCII

4.2.1 Archivos formateados columnas/filas sin encabezado

En este caso se puede asignar el contenido del archivo directamente a una matriz con la misma cantidad de filas y columnas que el archivo. Ejemplo:

```
OPENR, 1, ´mi_archivo.dat´
M = FLTARR(4, 3)
READF, 1, M
CLOSE, 1
```

Lo anterior supone que el archivo mi_archivo.dat estaba compuesto de 4 columnas y 3 filas.

Se usa OPENR cuando se desea abrir el archivo en modo lectura solamente. OPENW abre un archivo en modo escritura y si éste archivo no existe entonces se crea vacio, en caso contrario el archivo existente es vaciado. En primer argumento de estas funciones es un número que de ahí en adelante referencia al archivo abierto y debe ser el argumento de los procedimientos de lectura de tal archivo, tal como se mostró en el ejemplo. Una manera para que IDL busque automáticamente un número identificador libre es la siguiente:

```
OPENR, id, ´mi_archivo.dat´, /GET_LUN
...
FREE_LUN, id
```

Asi OPENR asignará un número identificador desocupado a la variable id y el procedimiento FREE_LUN liberará ese identificador después de haber sido ocupado y no se necesite tener más ese archivo abierto.

Ahora para escribir un archivo ASCII formateado, basta abrir el archivo con OPENW y escribirlo con PRINTF con el argumento del arreglo o variables que deseamos escribir en él.

Por ejemplo, supongamos que tenemos el arreglo X de 3 columnas y 8 filas:

```
OPENW, 1, ´mi_archivo.dat´
PRINTF, 1, ´Columna 1´, ´Columna 2´, ´Columna 3´
PRINTF, 1, X
CLOSE, 1
```

Nótese que en el ejemplo anterior hemos creado un archivo con un encabezado, por lo tanto no podríamos leerlo directamente con READF como hasta ahora hemos visto. Más adelante se verá cómo leer archivos con encabezado.

Si los arreglos a escribir a un archivo tienen muchas columnas IDL automáticamente cortará cada fila hasta un máximo de 80 caracteres. Esto algunas veces puede ser útil dado que muchos editores de texto no soportan líneas tan largas. Sin embargo la lectura del archivo se hace muy desordenada. Para superar este problema se puede

especificar el ancho (en caracteres) del archivo que deseamos crear, en los argumentos del procedimiento OPENW. Por ejemplo, si estimamos que nuestro arreglo X no tiene más de 1000 caracteres en cada fila, el siguiente código IDL imprimirá en el archivo test.dat el arreglo X tal como es (sin cortarlo en los 80 caracteres hacia el lado):

```
OPENW, 1, ´mi_archivo.dat´, WIDTH=1000
PRINTF, 1, X
```

La solución anterior supone que conocemos el ancho máximo en caracteres del arreglo X. Una solución más general entonces es especificar en cada escritura (PRINTF) el formato de cada fila, evitando que IDL escoja el formato por defecto. Por ejemplo:

```
nfilas = 40
ncols = 40
X = DIST( nfilas, ncols )
OPENW, 1, ´mi_archivo.dat´
FOR ifilas = 0, (nfilas - 1) DO $
    PRINTF, 1, X[*, ifilas], FORMAT=´(´+STRTRIM(ncols,2)+´F10.5)´
```

La función STRTRIM(*string, n*), quita de *string* los caracteres blancos o tabs que hayan en él dependiendo del número *n*. Si n = 0 los quita del comienzo del string, si n = 1 los quita en la parte posterior del string y si n = 2 entonces saca todos los caracteres blancos y tabs que hayan delante y detrás del string. Esto es conveniente de hacer cada vez que se pasa un tipo de dato (distinto de string) a string. Notemos que ncols no es string pero FORMAT necesita un string como argumento. Así, en el ejemplo anterior STRTRIM(ncols,2) devuelve 40 sin caracteres en blanco y la variable FORMAT entonces queda: ´(40F10.5)´, lo que significa que se escribirán 40 números de tipo Float, cuyo largo son 10 dígitos en total, de los cuales 5 son dígitos decimales[1]. Si el uso de STRTRIM asusta al lector principiante, entonces si se se sabe que hay 40 columnas se puede poner directamente:

```
PRINTF, 1, X[*, ifilas], FORMAT=´(40F10.5)´
```

Nótese que un * en las columnas del arreglo hace que IDL expanda todos los valores del rango del arreglo en columnas en esa misma fila. Por ejemplo:

```
PRINTF, 1, X[*, ifilas]
```

es lo mismo que:

```
PRINTF, 1, X[0, ifilas], X[1, ifilas], X[2, ifilas], ...  , X[ncols, ifilas]
```

Así, no es necesario especificar un segundo loop (ciclo FOR por ejemplo) para recorrer las columnas del arreglo.

[1] Esta es la notación típica de Fortran, que IDL hereda.

4.2.2 Archivos ASCII formateados con encabezado

Para leer archivos con encabezados de un número conocido de líneas se puede crear primero un arreglo de strings de tantas filas como líneas tenga el encabezado, luego leer el archivo (hasta el encabezado) para asignárselo a este arreglo y finalmente leer el archivo al arreglo de datos correspondiente tal como en la sección anterior. Por ejemplo supongamos un archivo llamado mi_archivo.dat como sigue:

linea 1 del encabezado
linea 2 del encabezado
linea 3 del encabezado
1.2 6.4 0.2
3.4 2.1 2.1
6.0 9.8 0.01
9.02 5.3 3.7

Entonces podemos leer los datos de la siguiente manera:

```
header = STRARR(3)
X = FLTARR(3,4)
OPENR, 1, ´mi_archivo.dat´
READF, 1, header, X
CLOSE,1
```

De esa manera la veriable header quedará con las líneas del encabezado y los datos numéricos quedarán en X.

Otra manera de leer un archivo con encabezado es con la función IDL READ_ASCII. Esta función tiene la opción de leer archivos dada una plantilla o template que puede ser creado interactivamente con la función ASCII_TEMPLATE. Esta función abre una ventana gráfica mostrando al usuario el archivo y dando la opción de seleccionar la línea del archivo en la cual comienza el conjunto de datos que interesa rescatar.

Ejemplo:

1. Leer un archivo con encabezado (mi_archivo.dat) sin template pero sabiendo que el conjunto de datos numéricos comienza en línea 5:
   ```
   data = READ_ASCII(´mi_archivo.dat´, DATA_START=5)
   ```
2. Leer un archivo con encabezado (mi_archivo.dat) con template consultando al usuario interactivamente:
   ```
   data = READ_ASCII(´mi_archivo.dat´, template=ASCII_TEMPLATE())
   ```

Hay que notar que READ_ASCII no devuelve un arreglo de datos, sino una estructura de vectores. Es decir, la variable data de los ejemplo anteriores no es un arreglo sino una estructura de arreglos que se pueden accesar de la siguiente forma:

```
PRINT, data.field1, data.field2, ...  , data.fieldN
```

dependiendo de la cantidad de columnas que tenga el archivo. Notar que cada data.fieldK (K entre 1 y N) es un vector que corresponde a cada columna del archivo leído, es decir se pueden recorrer en un loop los elementos data.fieldK[1], data.fieldK[2], ...

No es muy conveniente la forma de leer archivos del ejemplo 1 anterior, pues al no dar un template a READ_ASCII, esta función usa un template por defecto que no siempre se ajusta a la organización de los archivos que se están leyendo. Así, se recomienda usar templates creados con ASCII_TEMPLATE.

Si se tienen varios archivos con la misma organización (igual número de líneas de encabezado, igual número de filas y columnas) entonces no es necesario estar consultando al usuario cada vez por la forma que tiene el archivo (con ASCII_TEMPLATE) sino que se puede consultar sólo la primera vez, luego guardar este template en una variable y finalmente usarlo todas las demás veces para los otros archivos del mismo tipo que se deseen leer. Ejemplo:

```
t1 = ASCII_TEMPLATE(´mi_archivo.dat´)
data = READ_ASCII(´mi_archivo.dat´, template=t1)
data1 = READ_ASCII(´mi_archivo1.dat´, template=t1)
data2 = READ_ASCII(´mi_archivo2.dat´, template=t1)
etc...
```

Si el template se desea ocupar en otras sesiones IDL entonces también es posible guardar la variable template en un archivo. Ejemplo:

```
SAVE, t1, filename=´t1.sav´
```

en otras sesiones entonces se puede recuperar tal template de la siguiente forma:

```
RESTORE, ´t1.sav´
data = READ_ASCII(´mi_archivoN.dat´, template=t1)
```

Finalmente, la función DIALOG_PICKFILE es útil para que el usuario seleccione gráficamente el archivo que desee procesar. Su uso es el siguiente:

```
file = DIALOG_PICKFILE(filter=´*.dat´)
```

así la variable string file quedará con el nombre del archivo seleccionado por el usuario cuando éste presione el botón OK.

Capítulo 5

Rendimiento

Como IDL es un lenguaje orientado al arreglo, las funciones o procedimientos que operan sobre arreglos han sido optimizadas para entregar un mejor rendimiento que si se recorrieran con un loop aplicándoles los operadores correspondientes a cada elemento del arreglo. De la misma forma existen otras maneras para mejorar el rendimiento de los programas de acuerdo al orden que tienen los operandos en las operaciones. En este capítulo se verán algunos ejemplos de cómo mejorar el rendimiento de las aplicaciones IDL.

5.1 Ordenando de forma óptima los operandos

El orden de los operandos y operadores en una operación puede tener efecto en la rapidez de ejecución del programa por la cantidad de veces que se ejecuta la operación. Ejemplo:

```
B = A * 16 / MAX(A)
```

En la operación anterior suponemos que A es un arreglo de N elementos, entonces la operación anterior hace primero N multiplicaciones por 16 y luego N divisiones por el máximo elemento de A. En total la operación entonces tiene 2*N pasos. Sin embargo si cambiamos el orden de los operadores:

```
B = (16/MAX(A)) * A
```

entonces el mismo resultado se obtiene sólo en N + 1 pasos. El primer paso el división de 16 por el máximo elemento de A y luego multiplicación de los N elementos de A por el resultado de la división anterior.

5.2 Evitando IF dentro de loops

Como es suponer, un IF dentro de un loop es computacionalmente costoso debido a que la operación de comprobar la condición es ejecutada tantas veces como iteraciones tenga el loop. Normalmente se hacen loops para recorrer arreglos, así es posible aprovechar las características de IDL para el manejo de arreglos y evitarse en muchos casos algunos IF dentro de loops. Ejemplo:

```
FOR i=0, (N-1) DO $
   IF (A[i] le 0 ) THEN $
     A[i] = A[i] + B[i]
```

el loop anterior se puede cambiar por:

```
A = A + (B gt 0) * B
```

o bien:

```
A = A + (B > 0)
```

5.3 Usando funciones y operaciones optimizadas para arreglos

A continuación se mostrará el uso de dos funciones para arreglos. En los manuales de IDL se pueden encontrar otras como MIN, MAX, SORT, etc.

5.3.1 WHERE

retorna un vector de subíndices de un arreglo en los cuales los elementos asociados a esos subíndices cumplen con cierta condición dada como argumento a WHERE. Ejemplo:

```
FOR i=0, (N-1) DO $
   IF (A[i] le 0 ) THEN $
      C[i] = -SQRT(-A[i]) $
   ELSE $
      C[i] = SQRT(A[i])
```

el loop anterior se puede reemplazar por:

```
negs = WHERE( A lt 0 )
C = SQRT( ABS(A) )
C[negs] = -C[negs]
```

5.3.2 TOTAL

retorna la suma de los elementos de un arreglo o de una parte de él. Ejemplo:

```
sum = 0
FOR i=5, 60 DO $
   sum = sum + A[i]
```

el loop anterior se puede reemplazar por:

```
sum = TOTAL( A[5:60] )
```

5.3.3 Uso del *

Muchas veces no es necesario especificar loops en IDL pues si se quiere direccionar un arreglo en toda una de sus dimensiones se puede usar * en lugar de un loop. Por ejemplo:

```
FOR i=0, 511 DO BEGIN
  FOR j=0, 255 DO BEGIN
    temp = image[i,j]
    image[i,j] = image[i,511-j]
    image[i,511-j] = temp
  ENDFOR
ENDFOR
```

el loop anterior se puede reemplazar por:

```
FOR j=0, 255 DO BEGIN
  temp = image[*,j]
  image[*,j] = image[*,511-j]
  image[*,511-j] = temp
ENDFOR
```

Lo anterior supone que el arreglo image tiene 255 columnas, pues un * se expande en toda la dimensión sobre la cual es especificado.

5.4 Economizando el uso de memoria

Siempre un programa será más eficiente si usa menos memoria. Por esta razón ser conservador en el uso de la memoria siempre trae buenos resultados en lo que respecta del rendimiento en la ejecución de un programa.

Supongamos que tenemos un arreglo A muy grande. Si se quiere multiplicar ese arreglo por π entonces podemos hacer:

```
A = A * π
```

Lo que hace que IDL tenga en un momento de su ejecución dos copias del gran arreglo A (una con el valor antiguo de los elementos de A y otra con los valores nuevos de los elementos de A, para luego deshacerse de una copia, la antigua obviamente).

Una manera más eficiente de hacer lo mismo es:

```
A = TEMPORARY(A) * π
```

pues TEMPORARY no hace que IDL cree otra variable adicional para llevar a cabo el cálculo sino que usa la misma memoria que usaba el arreglo anterior.

Otra forma de ahorrar memoria es elegir el tipo de dato adecuado para cada variable que se cree. Por ejemplo si queremos usar una variable índice para recorrer un arreglo y este arreglo no tiene más de 255 elementos entonces usaremos el tipo de dato byte para nuestro índice.

Capítulo 6

Algunas funciones matemáticas de IDL

En este capítulo se mostrarán algunas funciones matemáticas de IDL. La brevedad de este capítulo no significa que IDL carezca de otras funciones matemáticas ni mucho menos, IDL posee una gran variedad de ellas. Al final de este capítulo se dan las funciones relacionadas con algunos temas que podrían ser de interés para el lector. Para más información consultar los manuales de IDL o la ayuda en línea.

6.1 Matrices

Como ya se ha visto a lo largo de este manual, IDL trabaja las matrices de la forma columna, fila. Por ejemplo:

```
A = [ [1,2,1], [2,-1,2] ]
```

$$\rightarrow \begin{pmatrix} 1 & 2 \\ 2 & -1 \\ 1 & 2 \end{pmatrix}$$

Para obtener la traspuesta de una matriz: T = TRASPOSE(A).

Para multiplicar matrices existen dos operaciones distintas, una de ellas hace la multiplicación por filas y la otra por columnas. Ejemplo:

```
A = [ [1,2,1], [2,-1,2] ]
B = [ [1,3], [0,1], [1,1] ]
PRINT, A # B
```

$$\rightarrow \begin{pmatrix} 7 & -1 & 7 \\ 2 & -1 & 2 \\ 3 & 1 & 3 \end{pmatrix}$$

```
PRINT, A ## B
```

$$\rightarrow \begin{pmatrix} 2 & 6 \\ 4 & 7 \end{pmatrix}$$

6.2 Ajuste de curvas (estimación de parámetros de modelos)

6.2.1 Estimación de parámetros de un modelo lineal

La función IDL para estimar parámetros de un modelo lineal es LINFIT. Véase el manual para mayor información. El uso de esa función es muy sencillo.

6.2.2 Estimación de parámetros de modelos no lineales

Dado un conjunto de datos (X_i, Y_i) y la fórmula general , f(x), de un modelo matemático con parámetros desconocidos, se pueden determinar éstos últimos minimizando un cierto criterio de error (mínimos cuadrados por ejemplo). El ajuste de superficies es el mismo pero el conjunto de datos es ahora (X_i, Y_i, Z_i) y la función f(x,y).

La función CURVEFIT se usa para hacer ajustes minimizando la suma de los cuadrados de los errores. Ejemplo:

Sea: f(x) = a (1 - e^{-bx}) ; a,b desconocidos.
X = [0.25, 0.75, 1.25, 1.75, 2.25]
Y = [0.28, 0.57, 0.68, 0.74, 0.79]

primero creamos una función IDL para evaluar f y sus derivadas parciales respecto de cada uno de los parámetros A_0 y A_1.

```
PRO funct, X, A, F, DPAR
  F = A[0] * (1.0 - EXP(-A[1]*X))
  IF (N_PARAMS() ge 4) THEN BEGIN
    DPAR = FLTARR(N_ELEMENTS(X),2)
    ; lo sig.  pone la der.  parcial respecto de A0 en la primera fi-
la de DPAR
    DPAR[*,0] = 1.0 - EXP(-A[1]*X)
    :  lo sig.  pone la der.  parcial respecto de A1 en segunda fila de
DPAR
    DPAR[*,1] = A[0] * X * EXP(-A[1]*X)
  ENDIF
END
```

En el comienzo de la función se pregunta por la cantidad de parámetros con que se llama a la función para ver si es necesario hacer el cálculo de las derivadas parciales. Esto depende entonces de como se llame a esta función.

Ahora calculamos A[0] y A[1] como sigue:

```
X = [0.25, 0.75, 1.25, 1.75, 2.25]
Y = [0.28, 0.57, 0.68, 0.74, 0.79]
W = 1.0/Y
A = [1.0, 1.0]
yfit = CURVEFIT(X, Y, W, A, SIGMA_A, FUNCTION_NAME=´funct´)
PRINT, A
```

$\rightarrow \left(\begin{array}{cc} 0.787386 & 1.71602 \end{array} \right)$

Así la función que acuerdo al modelo mejor ajustan los datos es:

y = f(x) = 0.787386 (1-$e^{-1.71602x}$)