SUN/50.24

Starlink Project
Starlink User Note 50.24

I D Howarth, J Murray, D Mills & D S Berry

5th March 2004

DIPSO — A friendly spectrum analysis program
V3.6-4
User Guide

Contents

1 Introduction

DIPSO is, historically, a simple plotting package incorporating some basic astronomical applications. If you just want to read in some data, plot them, and measure some equivalent widths or fluxes, you can do that without much effort. First-time users with this type of modest goal can skim through the documentation to get a feel for what's going on, then check the command reference section to find the commands required. You could even go straight to the terminal, and type `DIPSO`; there's no substitute for hands-on experience. However, DO read the documentation fully at some time; DIPSO can do a lot of things, some of which you might not know that you needed until you read about them....

While it is intended that simple things should be simple, an effort has been made to make complicated things possible. To this end, a number of rather rudimentary functions and free parameters are provided (with reasonable defaults set). A macro facility allows convenient execution of regularly used sequences of commands, and a simple FORTRAN interface permits "personal" software to be very simply integrated. The existence of this interface has encouraged the accretion of several codes for carrying out relatively elaborate numerical or astrophysical calculations (*e.g.* profile fitting, Fourier analysis, nebular continuum modelling). Because it has a monolith structure, DIPSO still runs fast, but there is, unfortunately, quite a lot of documentation to wade through to find the command you need. Still, you should persevere; somewhere, somehow, it is quite likely that DIPSO can indeed do what you want. (However, if you want to display images, or handle errors in a general way without doing a bit of coding, look elsewhere. You'll probably have to come back to DIPSO eventually in the latter case, though — and do a bit of coding!)

New features in this release, and some history related to earlier releases are contained in appendix B.

2 Getting Started

2.1 Absolute beginners

Sit down at a Starlink terminal, and type:

```
% dipsosetup
% dipso
```

You'll get a little "hello" message, and a new prompt:

```
>
```

Type

```
g9.z?
```

hit the return key. You will get an error message. Ignore this and read on. Type

```
Help, Q
```

and hit return. You have just completed your first DIPSO session, discovering on the way that DIPSO accepts more than one command on a line (each command being separated by commas), that upper- and lower-case inputs are accepted, and that DIPSO knows when you make mistakes (or at least, some kinds of mistake).

2.2 Doing something

You didn't do much, though; you'll need to know a few more commands. A full reference list of commands (ordered more or less alphabetically) is appended, but here we'll mention a few basic ones to get you going. (You should check the command descriptions for details of how they should be used.) Once in the program, you can use the `COMM` command to get lists of commands classified by function together with brief descriptions (for instance "`COMMANDS g`" will list all the graphics-related commands), or use `HELP` for more detailed information on individual commands. Data can be read in using the `READ` command, or in special cases one of the following commands: `ALASRD`, `SCREENRD`, `SP0RD`, `SP1RD`, `SP2RD`, `ATLASRD`, or `RESTORE`.

For "historical reasons" many people use the "Spectrum 0" format for input and output of data (`SP0RD`, `SP0WR`). However, the recommended file i/o commands are `READ` and `WRITE` (or `SAVE` and `RESTORE`), which preserve all the information which DIPSO associates with a data set.

To get a plotting surface, use the `DEV` command. Plotting is usually done with `PM`; unless you've provided X and Y ranges (with `XR` and `YR`, or some combination of `XMAX`, `XMIN`, `YMAX`, and `YMIN`) the plot is auto-scaled to the minimum and maximum values in the arrays.

Once you have managed to read in some data, and plot them, you will soon want to carry out measurements, change the style of the plots, and so on. To find out how to proceed, you should read the descriptions of commands like (`HIST`, `POLY`, `MARK`); (`XV`, `YV`, `XYV`); (`CSET`, `CROT`); and (`TPORT`, `TZONE`).

Type `q` to leave the program. If in the middle of something long and tedious you despair, you can type control-C; this stops execution of the current command, and returns you to the DIPSO command prompt.

If you decide in the middle of a DIPSO session that you need to issue some operating system commands, then simply push the DIPSO task into the background by typing control-Z, issue your operating system commands, and then re-enter DIPSO by typing `fg`.

3 Data Storage

3.1 Internal Data Storage

On being read in, data are stored in the 'current' X,Y arrays, which have space reserved for up to 200,000 pairs of points. By default — or, in some cases, by compulsion — most operations (*e.g.* plotting) are carried out on data stored in these arrays. Data can be saved for later use by '`PUSH`'ing them onto a 'STACK', which can be thought of as a series of X,Y arrays. The STACK contents can be inspected using `SL` ('Stack List'), deleted using `DEL`, and brought into the 'current' arrays using `POP`. Up to 200 stack entries, or 800,000 points, are allowed.

3.2 Data Storage on Disk

The contents of the current array can be written to a disk file using the WRITE command, and the contents of the stack can be written using the SAVE command. These files can either be standard Starlink "NDF" structures (see SUN/33), or alternatively files containing unformatted data in the original DIPSO format used prior to version 3.00.

The USENDF command allows the user to select which format to use. All commands which read or write data to or from the current arrays or stack are influenced by the USENDF command setting unless the description of the command in appendixA says otherwise.

Use of NDF structures enables data files created by DIPSO to be used by other Starlink packages (and vice-versa). It also enables data files to be transferred freely from one operating system to another without needing to do a format conversion for each one. NDF structures are contained within disk files which have the file extension ".sdf". When referring to an NDF, *do not include the file type or an error will result*. Think of it this way; an NDF is an object contained *within* a disk file. The NDF and the disk file in which it is contained are separate entities and can in principle have different names. When asked for an NDF you give the name of the NDF, *not the name of the disk file*. It just so happens that at the moment an NDF named "my_data" will be contained in a disk file called "my_data.sdf" but this may not always be the case.

In addition, columns of values can be read from FITS binary or ascii tables into the current arrays.

4 Command Input

DIPSO is basically command driven, although for some of the more complex algorithms the program prompts on a step-by-step basis. Many commands can be input on a single line (in upper or lower case), each command (with its associated parameters) being separated from its neighbours by a comma. Parameters associated with a particular command follow it on the command line, separated by spaces. Parameter values which include commas and/or spaces need to be enclosed in double quotes when given on the command line, otherwise the commas and spaces will be interpreted as delimiters. Any mandatory parameters not specified with a particular command are prompted for, and failure to complete a command will generally result in any remaining commands on the line being ignored. *e.g.*, the line:

```
READ TEST, DRED, PM, PUSH
```

will read in the NDF test from the disk file test.sdf. If DIPSO fails to read the NDF successfully, you get an error message and the remainder of the line is ignored. Otherwise, it will attempt to de-redden the data using a 'standard' extinction law (DRED). Since a value of E(B-V) is mandatory for this command, but has not been provided, it is prompted for:

```
DRED: E(B-V)?
```

(Similarly, if READ hadn't been told which NDF to read, this parameter would have been prompted for.) On provision of the appropriate number, the data are de-reddened, plotted

(PM) on the (previously assigned) plotting device, and then PUSHed onto the STACK. (If a plotting device were not previously assigned, DIPSO would again report an error and terminate execution of the command line.)

If you're letting DIPSO prompt you for mandatory parameters, and decide that you want to abort the command line, you can respond to the parameter prompt with one or two exclamation marks (e.g. `!` or `!!`). In the first case, the current command will abort, the remainder of the command line will be rejected, and you will be returned to the DIPSO command prompt. In the second case, the current command will abort, and DIPSO will also abort (saving the stack to EXIT_STK.sdf or EXIT.STK in the process), returning you to the operating system.

Some commands have optional parameters in addition to any mandatory ones. For example, DRED has three associated parameters:

```
DRED E(B-V) R MODE
```

of which E(B-V) is the only mandatory one, and therefore the only one prompted for if not supplied. The other two parameters are R (= A(V)/E(B-V)) and MODE, a switch which allows an LMC-type extinction law to be invoked. Optional parameters have defaults supplied; in this case, R=3.1 and MODE=0 (Galactic law). If you want an LMC-type law with R=3.1 you must provide all parameters:

```
DRED 0.5 3.1 1
```

but if you want a Galactic law with R not equal to 3.1 you only need type (*e.g.*)

```
DRED 0.5 2.0
```

If you provide too many parameters, the command will use those it can, issuing a warning about those it can't; *e.g.*

```
DRED 0.5 3.1 0 99.99
```

will provoke a warning that redundant parameters have been provided.

4.1 Command Line Recall and Editing

Command line recall and editing is available in DIPSO. Use the up/down arrows or the RECA command to recall commands, and the left/right arrows and the delete/backspace key to edit them. The following control characters are recognised:

Control-A : Jump to the start of the input buffer

Control-E : Jump to the end of the buffer

Control-N : Toggle overstrike/insert mode

Control-U : Empty the input buffer

Two separate lists of text strings are kept; one for the command lines given in response to the DIPSO command prompt, and one for the strings given in response to the prompts issued by each command. The up and down arrows, and the RECA command operate within each list, independently of the other list.

5 Command Procedures

Commands can be input from macros (script or command files). This can be particularly useful if you frequently carry out a fixed sequence of operations. The command file can be in the directory from which you are running DIPSO, or in a default directory (OWNERDIR) of your own assignment. Thus if DIPSO is requested to execute a command file (without a full directory specification being given) it first looks in the current directory; if it doesn't find it, it looks in a directory assigned the environment variable OWNERDIR; and if it still doesn't find it, you get an error message. All your frequently used command files can therefore be kept in one place. For example, a file called TEST.CMD may contain the instructions:

```
READ,DRED
LOGY,YMULT -2.5,XMULT 1.0E-04,XINV
PUSH,SL
```

The commands in this file would be executed by typing:

```
@TEST
```

The unspecified mandatory parameters for READ and DRED would be prompted for, and input, at the terminal. The Y values in the 'current' array would be replaced by Log(10) Y (LOGY), then multiplied by -2.5 (YMULT -2.5); the X values would be multiplied by 10^{-4} (XMULT +1.0E-04), then replaced by $1/X$ values (XINV). The final data would then be PUSHed onto the stack, the contents of which would be displayed at the terminal (SL).

On completion of the commands in the file, control returns to the terminal.

On startup, DIPSO looks for a command file called startup.cmd in a directory which has been assigned the environment variable OWNERDIR. So, if you regularly want to change any default settings from those normally set, just create such a file, containing commands which will set your customised options (*e.g.* you may not like the standard X and Y labels, or you might want always to use native DIPSO data files rather than NDF data files, *etc.*).

6 Batch Processing

To run a dipso job in the background place all the commands you wish to run in the startup.cmd command file and then type:

```
% dipso &
```

at the shell prompt. If you wish to direct the output to a log file then type:

```
% dipso >mylog &
```

In addition you can use the UNIX input/output redirection operators $>$ and $<$ to direct commands into DIPSO from other files or even from other programs (see UNIX reference manuals for details).

7 Plotting

The plotting commands sit on top of the GKS/SGS/AUTOGRAPH packages (see SUN/83, SUN/90, *etc.*; sometimes the command names don't relate in an obvious way to the name of the graphics routine which is called, because DIPSO was originally written using a different graphics package). Although DIPSO grew with simple data sets in mind (*i.e.* monotonically changing X values) it will plot some more complex arrays. (The example program for the user interface, described below, generates a circle.)

7.1 Plotting options

Plotting can be done with a variety of symbols (`MARK`, `MROT`), or line types (`TLINE`, `TROT`) in `POLY` (*i.e* join-the-dots) or `HIST` (histogram) mode. If you have access to appropriate hardware, colour plotting is also possible (`CSET`, `CROT`). Device changes can be made at any time, so that you can, for example, switch between an Ikon, Pericom, and laser printer at will. Alternatively, you can stick to a single device and display data in different zones of the plotting surface. A set of useful sub-zones is provided automatically (see the `TZONE` command).

7.2 Cursor commands

An important aspect of any plotting package is making measurements from, or marking points on, a plotting surface using a cursor (where available). In DIPSO, the cursor will respond to any alphanumeric key. If the functionality of the command requires only one cursor hit (*e.g.* `XV` to measure X values), then the command is exited by making two cursor hits at the same point. This method of exiting generalises to other cursor-driven commands which require multiple inputs (*e.g.* `CREGS`).

7.3 Default plotting (and other) options

The default options (all of which can be changed at will) are:

`DEVice 0` - (null device)
`HIST` - ("histogram" plotting style)
`TZONE 0` - (use the entire plotting surface)
`NXY` - (auto-scaling on X and Y axes)
`CSET 1` - (plot in white on the Ikon)
`TLINE 1` - (continuous lines)
`BOX` - (clears frame between plots)
`NOFILL` - (MARK symbols open)
`TICKS <null>` - (Tick marks on axes calculated automatically)
`FONT 0` - (Hardware character set)
`XJ`, `YT` - (Plot has "justified" X axis and "trimmed" Y axis)
`LABON` - (Full labelling of axes)
`GRIDSTYLE 1` - (Four sides to the plot box drawn in)
`PPROMPT F` - (PM without arguments plots current arrays)
`XLAB "Wavelength"`

```
YLAB "Flux"
```

These defaults are chosen as a compromise between aesthetic elegance and speed of plotting. For an ugly but fast plot, choose POLY and GRIDSTYLE5; for truth and beauty, choose FONT 2.

This is as good a point as any to note some other default settings for DIPSO:

ECHO -1 - (Commands file inputs not echoed at the terminal)
BEEP - (input errors induce a beep)
HANDLER 1 - (robust error handling)
TPROMPT F - (doesn't insist on a string with TITLE)
USENDF T - (NDF structures are used to store data on disk)
USEHTX F - (Help information is displayed in plain text format)

7.4 Getting hardcopy plots

DIPSO doesn't "remember" what is on the plotting surface in any way. Thus you can't get an "instant" copy of a plot on your terminal (unless you have some special hardware which will do it for you). Instead, you must change devices and execute the appropriate series of commands to do the plot for you.

Plotting on a hardcopy unit (laser printer, line printer, *etc.*) will normally leave a (frequently large!) file in your working directory, and this file will need to be printed on the appropriate device before you actually get a plot out. Check the GKS documentation (or your node manager!) for details; and remember to tidy up your directory afterwards, or you will quickly run out of disk quota!

8 The User Code Interface

If DIPSO can't or won't do something reasonably straightforward (*e.g.* a simple functional operation on a single spectrum) — or even, if you're ambitious, something quite complicated — that you require of it, then you can avail yourself of the user interface. This consists of a logical function, USER, which gives simple and straightforward access to the contents of the 'current' arrays; and two user-callable subroutines, UPUSH and GETSTK, which allow you to respectively get data from and put data onto the stack. (These two routines are described at the end of this section.) You are also free to use any GKS, or SGS routines you may need.

If you want to do something that requires opening new input streams, it is recommended that you use streams 23-29 inclusive. DIPSO closes most i/o streams as soon as it has finished with them, but it always has stream 22 open, and you are strongly discouraged from using streams 5 and 6 for anything other than standard input and output (*i.e.* the designated device; normally the terminal when DIPSO is used interactively). (The reason that stream 22 is always open is that GKS sends its error messages to this stream. Such messages are more likely to annoy than enlighten DIPSO users!

8.1 The "LOGICAL USER" Function

To use the interface you will need to write a subprogram called `USER` which should follow the example given in `$DIPSODIR/user.f`.

The example subroutine contains almost all the additional documentation needed to understand how to 'do your own thing'. This documentation may seem a bit technical, but don't be put off by that (consult one of your local computer devotees if in doubt); the interface is really *very* easy to use. As a bare minimum, you could copy the example subprogram and just add in your own IF block:

```
ELSE IF( CMD .EQ. '<your command>' ) THEN
   <carry out operations>
```

The `USER` function delivers lots of variables for you to play with, but the essential contents of the argument list are:

- the current X and Y arrays
- the current command name
- a string, PARAMS, containing the parameters associated with the command.

Sometimes you will want to treat `PARAMS` as a character string (for example, it might be a filename); but more often you will want to read numbers from it. You can do this in exactly the same way as DIPSO does by calling the routine `DECODE`:

```
CALL DECODE( CMD, PARAMS, NP1, NP2, VALUES, PROMPTS, OK )
```

where:

CMD - character The command name, passed to `USER` by DIPSO.

PARAMS - character Associated parameters, passed as a string by DIPSO.

NP1 - integer The minimum number of parameters (between 0 and 10).

NP2 - integer The maximum number (greater than or equal NP1).

VALUES - real The array into which real values, decoded from PARAMS, are passed.

PROMPTS - character The NP1 prompts for mandatory parameters.

OK - logical A success/failure switch.

The '`PROMPTS`' *must* be left justified, separated by blanks, and terminate with a blank. (Have a look at the `$DIPSODIR/user.f` code for examples of how to use `DECODE`.

8.2 Building your own binary

Having written your USER code, you can build your own dipso binary.

You must first set the environment variable SYSTEM appropriately for your system: alpha_OSF1 for AXP OSF/1, or sun4_Solaris for Sparc Solaris 2.x.

Then, you can compile and link a new binary by typing:

```
% dipsosetup
% my_dipso
```

If you wish to link in subroutines contained in object modules sub1.o, sub2.o etc, you should define an environment variable MY_OBJECTS specifying these object files:

```
% setenv MY_OBJECTS "sub1.o sub2.o sub3.o"
% dipsosetup
% my_dipso
```

This leaves you with a personal copy of dipso in the directory you're working in. You should then set up an alias to run this version in preference to the system version:

```
% alias dipso <where ever>/dipso
```

8.3 Debugging your code

(This section can be skipped by people who never make programming errors!)

DIPSO is equipped with a condition handler to prevent crashes. DIPSO shouldn't give a crash in the normal run of things (if you get one, please report it, giving fullest details possible — preferably a macro file which always results in the crash), but it may well do so in user-supplied code. In this case, you will normally want to disable the condition handler in order to get the system handler, which may tell you where the crash occurred. To turn the handler off, the HANDLER command can be invoked (use HANDLER 0). On some flavours of UNIX, this may result in the program appearing to freeze after a crash. If this happens, pressing control-C should return you normally to a system prompt.

8.4 Local documentation

If your site has a 'user-enhanced' version of DIPSO that is used by several people, then it might be convenient to put the executable into a local public directory. In this case you should persuade your node manager to create such a directory, with appropriate protections, and give it the environment variable LDIPSODIR. The local version can then be run, of course, as $LDIPSODIR/dipso.

In response to the COMM command, DIPSO *first* looks for a file $LDIPSODIR/command.hlp, which will be used if found. It will then go on to look for a file $DIPSODIR/comand.hlp which it will also use if found.

So it's possible to put an appropriately modified version of `command.hlp` into `LDIPSODIR` to keep users informed of local additions to available commands. The local version of `command.lis` should contain only the extra local commands, not the standard commands, and should be formatted like the standard version.

Note, in previous versions of DIPSO, command information was stored in a file called `command.lis` which had a different format to `command.hlp` files and was processed differently. It is recommended that any local `command.lis` files still existing be converted into `command.hlp` format (see `$DIPSODIR/comand.hlp` for a description of the format). If DIPSO finds a local old-style `command.lis` file it will simply display its contents (as in previous versions of DIPSO) in preference to using any new-style `command.hlp` files. Note, however, that the extended functionality of the `COMM` command (i.e. word searches, command descriptions and classification) will not be available.

It's possible to give `HELP` for local commands, too, although it can only be accessed as plain text (see `USEHTX`). The `HELP` command in plain text mode first runs through `$DIPSODIR/help.lis` but if it doesn't find a command name there, it will try to look for an `$LDIPSODIR/lhelp.lis`, and search that for help information. This file should match the format of `$DIPSODIR/help.lis`, but need contain information only on local commands.

The first thing that DIPSO does is look for a file called `$LDIPSODIR/updates.lis`, and print out anything it contains. So if you've made changes, you can announce them to your local community through this mechanism.

8.5 Data access

If you want to do complex operations involving several data sets, you may well want to access data on the DIPSO stack. Well, you can; but first, you'll need to understand a bit more about how DIPSO stores data.

8.5.1 More on data storage

A DIPSO data set contains a variety of information. First of all, there is a brief header string [CHARACTER*80 TITLE]. Then, of course, there are the X and Y data arrays [X(MAXPT), Y(MAXPT), MAXPT=64000], which contain the NPOINT pairs of data points. Now, in order to know where in the data any gaps occur, DIPSO maintains a separate 'break' array [BREAKS(MAXBRK), MAXBRK=1000] which contains the NBRK break points associated with the data set. A break point is the index, in the X and Y arrays, of the last point before a gap in the data set. Thus if there are 200 points in the data set, and there are breaks between the 7th and 8th, and 123rd and 124th, data points, then BREAKS(1)=7, BREAKS(2)=123, and BREAKS(NBRK)=200, where NBRK is 3. Note that the last point in a data set is always a break point, so that NBRK is always 1 or greater.

To allow compatibility with some other programs (notably IUEDR), DIPSO assumes that a specific Y value (zero by default) actually flags a gap in the data, for some i/o commands (*e.g.* `SPORD/WR`). This will often be invisible to the user, but you ought to keep it in mind. Note also that if DIPSO reads in a data set where a gap is padded out with a whole string of zeros (*e.g.* from IUEDR), then it throws away all but a couple of them, to save space. (Try `SPORD`ing a hi-res IUEDR spectrum, then `SPOWR`ing it; the output is much smaller than the input). This behaviour does not apply to the `READ`, `WRITE`, `SAVE` and `RESTORE`.

When using NDFs, there is another Y value which is also used (by *all* commands which access NDFs) to flag gaps in the data. This value is the standard Starlink "bad" value which is used to flag invalid or missing data in many other Starlink packages. Its value is -1.7014117E+38 (software generates these "bad" values automatically... you're not expected to type them in!).

Finally, although DIPSO will plot general X,Y arrays, several of the applications commands expect and require data that have Angstrom or km/s as the X unit (*e.g.* `EW`). A variable, WORV (which means "Wavelength OR Velocity"), is used to flag data in which the "X" unit is km/s; if this is the case, then WORV=λ/c, where λ is the rest wavelength to which the velocities are referenced (in Angstroms) and c is the speed of light (km/s). Otherwise, WORV=1.0. (You'll just have to think carefully about what you're doing if your data are in frequency units, I'm afraid — WORV=1.0 will generally be associated with your data.)

8.5.2 Creating NDFs in your own programs

To output data from other programs in a form suitable for inputting to DIPSO with the (recommended) `READ` command requires the following minimal code (with appropriate values and names for all variables):

```
*  Global Constants:
       INCLUDE 'SAE_PAR'         ! Include standard starlink constants
                                 ! such as SAI__OK.

*  Local Variables:
       CHARACTER  COMM           ! This should be WRITE or SPOWR and
                                 ! causes the corresponding command to
                                 ! be simulated.
       CHARACTER  NDFNAM         ! Name of output NDF structure. NB, don't
                                 ! include a file type!!
       INTEGER    NPOINT         ! Number of points in XV and YV.
       REAL       XV( NPOINT )   ! X axis values.
       REAL       YV( NPOINT )   ! Y axis values.
       CHARACTER  XLAB           ! X axis label.
       CHARACTER  YLAB           ! Y axis label.
       CHARACTER  TITLE          ! NDF title.
       INTEGER    NBRK           ! Number of points in `breaks` array.
       INTEGER    BREAKS( NBRK ) ! Breaks array.
       REAL       WORV           ! Wavelength of velocity parameter.
       INTEGER    STATUS         ! Should be SAI__OK on entry. SAI__OK on
                                 ! exit if successful.

*  Create the output NDF.
       CALL WRITE_NDF ( COMM, NDFNAM, NPOINT , XV , YV , XLAB, YLAB,
     :                  TITLE , NBRK , BREAKS , WORV , STATUS )
```

Such programs then need to be linked with the DIPSO object libraries, and the NDF subroutine library (see SUN/33). To create a program called `fred` which uses `WRITE_NDF` to create an output NDF, do the following:

```
% star_dev
% dipsosetup
% f77 -o fred fred.f -L$DIPSODIR -ldipsot -L/star/lib `ndf_link`
% star_dev remove
```

As well as being readable by DIPSO such output data sets will also be automatically readable by all the standard STARLINK packages on any of the supported operating systems.

8.5.3 Getting data from the stack

So, now you know what's in a DIPSO data set; and thus, you have a good idea of the information on the stack. DIPSO lets you take copies of stack data using calls to the subroutine GETSTK:

```
      CALL GETSTK(INDEX, NPOINT, XV, YV, NBRK, BREAKS, TITLE,
     :            WORV, OK)
```

where:

INDEX - integer The stack entry you want to access.

NPOINT - integer On calling, the size of the arrays into which the XV and YV data are to be loaded; and on exit is the number of elements of the arrays which are occupied (*i.e.* the number of points).

XV - real A user-supplied array, which contains the X values of the STACK entry on return. (It is your responsibility to ensure that the array is big enough to hold all the data from the STACK entry.)

YV - real A user-supplied array to hold Y values on return.

NBRK - integer On entry, NBRK is the size of the BREAKS array. On return, NBRKS contains the number of 'break points' in the data set. (Again, you must ensure that enough space is available.)

BREAKS - integer A user-supplied array of length NBRK, to hold the indexes of 'break points' in the XV array.

TITLE - character The title associated with the data set.

WORV - real Wavelength *or* Velocity.

OK - logical Success/failure flag. OK = .FALSE. if the call to GETSTK is identified as unsuccessful.

8.5.4 Pushing data onto the stack

You can also push data onto the stack:

```
      CALL UPUSH( ASZE, XV, YV, NPOINT, BSZE, BREAKS, NBRK, TITLE,
     :            WORV, OK)
```

The arguments are the same as for GETSTK, except that ASZE (integer) is the size of your arrays holding the X and Y values, and BSZE (integer) is the size of your BREAKS array.

To encourage you to look on DIPSO as a tool with which you can interface your own software, it is worth noting that the ELF package (described below), all the Fourier analysis software, the IS routines, and the NEBCONT facility were added to DIPSO with very little more than the basic interface described above.

If you do write some software that you think may be of general interest, please contact Ian Howarth (idh@star.ucl.ac.uk); it may be possible to incorporate it into the public version of DIPSO.

9 Emission Line Fitting (ELF)

DIPSO has access to a suite of subroutines which are designed to fit a variety of line profiles to observed data. The commands really need a bit more explanation than can readily be put into the alphabetical reference list which follows; so here's a bit more explanation...

The primary purpose of the ELF routines is to separate blends by fitting multiple profiles. Gaussian profiles are the most commonly used, but other analytic forms are possible, as are 'numerical' profiles. Facilities are provided for constraining line centre positions, widths, and relative fluxes, so that known atomic data (such as relative wavelengths or intensities within multiplets) can be utilised. The option of relative flux constraint is particularly useful when analysing optically thin emission lines (hence Emission Line Fitting), but the package is entirely happy with absorption lines (which it treats as emission lines with negative fluxes), or any other form of data that can be reasonably approximated with the available profile forms.

Fits are made to the spectrum data stored in the DIPSO 'current' arrays. The continuum level may be set to zero by manipulations within DIPSO, or a polynomial fit to the continuum can be made simultaneously with the profile fitting. (The former option is *strongly recommended*.) Specification of constraints and starting values for the fit parameters is done in a command language (invoked by the DIPSO command ELFINP) which is described below. After optimisation of parameters (DIPSO command ELFOPT), the full specification of the fit and the results may be stored. The resulting fit, in spectrum form, can be pushed on the DIPSO stack (ELFPUSH).

9.1 ELF commands

The complete set of ELF commands are described in detail in appendixA. They are:

```
ELFINP   ELFOPT   ELFNEWC
ELFPUSH  ELFLFIX
ELFPUSHC ELFPOPC  ELFDELC ELFCSL ELFVUC
ELFSAVEC ELFRESTC ELFWRC
ELFPIN   ELFPL
```

As you can see, they are all of the form ELF. . . , so that they can easily be found in the documentation.

9.2 ELF data storage

The ELF package takes a COPY of the DIPSO 'current' spectrum. The maximum space available is 1000 datum points. This number is deliberately rather smaller than the space available in DIPSO itself, since fitting to large numbers of datum points is prohibitively time consuming.

Fit data are maintained in three storage areas:

- A 'current' area (not to be confused with the DIPSO current arrays) contains the specification of the fit in progress. If an optimisation has been carried out, the results (in the sense of optimised coefficients) are also kept in this 'current' array.
- A stack of fit coefficients. Data may be interchanged between this stack (again, not to be confused with the main DIPSO stack) and the ELF 'current' area. Space is provided for a maximum of 20 lines in each fit.
- A stack of input numerical profiles. These *must* be spectrum data without internal breaks (gaps), stored in VELOCITY space.

The stack of profile types has space for up to ten entries. The first five of these are reserved for analytically specified profiles, and the last five for numerical profiles. Profile definition is as follows:

Profile 1: Gaussian. (C=centre, W=width(FWHM), I=peak flux)

Profile 2: Triangular. (C=centre, W=width(FWHM), I=peak flux)

Profile 3-5: Unused at present.

Profile 6-10: Available for numerical profiles.

9.3 ELF general procedures

Although the command descriptions given later describe functionality in detail, it is probably worth just summarising how to do a simple Gaussian fit, for illustration. The steps would typically be:

- POP the data of interest into the current arrays. Use (*e.g.*) RXR to restrict the number of data points to the minimum consistent with adequately defining the line(s) of interest and a small amount of continuum.
- Although you can represent the continuum by a polynomial with free parameters, convergence is enormously improved if you first subtract a continuum (using PF, or CDRAW, for example, together with ASUB). If you really must incorporate a background polynomial in the fit, keep the degree as low as possible (*e.g.* zero) to avoid indeterminacy.
- Type ELFINP to invoke ELFs special command language, and input your first guesses at the values of the parameters to be optimised. Make life easier for yourself by keeping the number of non-linear parameters (line centre positions and widths) as small as possible, and try to make your guesses good ones. Type QELF to leave the command language processor.

- Type ELFOPT to start the ELF Fit Coefficient OPTimisation. If you have many free parameters, and the machine is being heavily used, go for a cup of tea, after you've checked the first iteration of the optimisation to make sure everything is as you expect.
- Later... you can push a copy of the 'best fit' model onto the DIPSO stack using ELFPUSH. You can also save the fit coefficients on the separate fit coefficient stack, for later use or reference, with ELFPUSHC.

A serious program crash while optimising, such as divide by zero or overflow, may occur occasionally. Such problems are usually caused by overspecified fits, or starting values that are grossly in error.

10 Words of Warning!

Of course, DIPSO generally does nothing that you don't ask it to do. So, if 'nothing' happens, it is probably because you've defined X and Y ranges that exclude the data, or left the BOX switched off (NB), or you're not plotting on the device that you think you are. Something else to watch out for is plotting a STACK entry, and then trying to do some operation on the plotted data instead of the data in the 'current' arrays. Be careful!

DIPSO carries no 'memory' of what's on a particular plotting surface. This means that if you want a laser-printer plot of what's on the screen you need to change device and actually do the plot again (don't forget to change back to your graphics terminal when the plot is finished!).

11 Acknowledgements

Several of the more elaborate computational functions in this version of DIPSO have been grafted on as a result of people exploiting the user interface, with their code eventually being adopted for the release version. The biggest single contribution is the ELF package, which was developed by Pete Storey (pjs@star.ucl.ac.uk). Pete also had a hand in NEBCONT, which uses code primarily written by Pat Harrington (U. of Maryland). Stephen Boyle (sjb@star.ucl.ac.uk) donated the Fourier, periodogram, and cross-correlation routines; the interstellar line profile code has a long and chequered history, but was first brought forth in the good old days of punched cards by Clive Davenhall. DIPSO's basic structure and command interface owes much to Dane Maslen. Other contributors include Jack Giddings, Des Middlemass, David Monk, and Starlink Management.

A Command Specifications

A.1 Individual Commands

This section contains a roughly alphabetical reference list of commands, with a description of the the actions invoked. (Strict alphabetic ordering has been sacrificed in one or two places in order to group together the texts for closely related commands.) Each command has its associated parameters listed with it, in the order in which they must be supplied. Optional parameters (for which defaults are provided) are given in [brackets].

@ (AT) *filename[.typ] p1 p2 p3 ..... p9*

Reads commands from a command file. Any prompts for unspecified mandatory parameters are given at the terminal. Command files can include blank lines and 'comment cards'; the latter must have an exclamation mark [!] or asterisk [*] as the first character. (Comments may not be flagged with a first-column "C", because this could easily be the first character of a command.)

On successful completion (or on failure to execute a command) control returns to the terminal.

IMPORTANT: Command files may not contain references to other command files (nor to themselves), for fairly obvious reasons.

DIPSO first searches for a file of the given name in the current directory (or whatever directory is given in the file specification). If it fails to find it, it then looks for a file `$OWNERDIR/<filename>`. Thus you can keep a set of frequently used command files in the directory assigned the environment variable `OWNERDIR` (this assignment would normally be carried out in your login scripts).

The default file type ([.typ]) is '.cmd'.

AADD *n*

Adds the contents of the 'current' Y array to the values in STACK entry 'n', leaving the result in the 'current' array. Both data sets must have monotonic X arrays for sensible results to emerge. To perform the arithmetic, the data in the 'current' arrays are mapped onto the STACK X grid. The addition is only performed at X values where there are valid Y values in both data sets. In conjunction with `GRID`, the `AADD` command can be used to remap data.

ADIV *n*

Divides the Y values in STACK entry 'n' by the Y values in the 'current' arrays. In order to do this, the 'current' data are mapped onto the X grid of the STACK data (both X grids must be monotonically changing). The output data are left stored in the 'current' arrays.

If both data sets are recognized by DIPSO as having X units in velocity space (WORV not equal 1; see the `TOV` command for details), then the output data are corrected by the ratio of the WORVs (i.e. by the ratio of the central wavelengths) to give true flux ratio as a function of velocity. If only one data set is recognized as having X units of velocity, the division is carried out and an error is reported.

AMAX ***n***

At each X point, puts the larger of the STACK (entry 'n') and 'current array' Y values into the current array. To do this, the data in the current arrays are mapped onto the STACK X grid. The function returns Y values only at those X values where valid data occur in both data sets. AMAX may be useful, when used with AMIN, for displaying the envelope to a set of spectra of a given object.

AMIN ***n***

At each X point, puts the smaller of the STACK (entry 'n') and 'current array' Y values into the current array. To do this the data in the current arrays are mapped onto the STACK X grid. The function returns Y values only at those X values where valid data occur in both data sets. (See also AMAX).

AMULT ***n***

Multiply the Y values in STACK entry 'n' by the values in the 'current' arrays. The data in the 'current' arrays are mapped onto the X grid of the STACK data in order to carry out this operation. (The X grids are both required to be monotonic). Results are left in the 'current' arrays.

ASUB ***n***

Subtracts the 'current' Y array from STACK entry 'n'. In order to carry out this operation, the 'current' data are mapped onto the X grid of the STACK data. Subtraction is only carried out at X values where there are valid Y data in both STACK and current arrays, and both X arrays must be monotonic. Results are left stored in the 'current' arrays.

ASWAP ***(no parameters)***

Swaps the 'top' (*i.e.* numerically largest) STACK entry with the contents of the 'current' arrays.

ALASCHK ***(no parameters)***

Checks the current defaults for the lines and columns to be read in using ALASRD.

ALASCOLS ***xcol ycol***

Tells ALASRD to read X and Y data from the specified columns of a file. (A "column" is a string of alphanumeric characters separated from other columns by spaces; of course, the columns which ALASRD actually acquires must contain exclusively numeric values.) When DIPSO begins, these are set to 1 and 2 by default; they are *not* reset to these values after execution of ALASRD.

ALASLINS ***line1 line2***

Tells ALASRD to read only lines line1 to line2 (inclusive) of an input file. If line2 is specified as zero, this is interpreted to mean end-of-file.

ALASRD *filename[.typ] [brkval]*

Reads data from a formatted file. The simplest structure which ALASRD (and DIPSO) can read is one pair of X, Y values per record; this is the default file structure expected by ALASRD. More elaborate files can be read in through prior use of the ALASLINS and ALASCOLS commands (q.v.).

Gaps in the data are assumed to be flagged by Y values of zero, unless a different "brkval" is specified on the command line.

The default file type ([.typ]) is '.DAT'.

ALASWR *filename[.typ] [brkval]*

Writes the contents of the 'current' arrays into a formatted file. The X data are output in column 1, and the Y data in column 2; gaps in the data are flagged with a Y value of zero, unless a different value for "brkval" is specified on the command line.

The default file type ([.typ]) is '.DAT'.

ANGLE *Theta*

Changes the angle at which PWRITE strings and MARK symbols are plotted. Theta is measured in degrees, anticlockwise from the horizontal, and is initially set to zero.

ATLASRD *Teff LogG [MODE]*

Reads in Kurucz model atmosphere fluxes, from the data base kept in the directory with environment variable SPECDAT. The data are stored in the database in the form of astrophysical fluxes; however, ATLASRD multiplies them up by a factor π to produce 'actual' fluxes. The x unit is Angstroms, and the y unit erg/cm2/s/A.

If MODE=0 (the default value) solar abundance models are acquired; otherwise the 'low metal abundance' models (1/30 solar) are read in. A summary of the available solar abundance models can be obtained using ATLIST, and the models normalised to cursor-selectable X,Y values using ATNORM.

Other files in the SPECDAT database, including extended atmosphere and Non-LTE models (see $SPECDAT/info.lis for details), can be read in using SP2RD $SPECDAT/filename.typ. The KHMEXT, MLTE and MNLT models can all be ATNORMed if they are first subjected to TENY, but the normalisation constant will be meaningless.

If you cannot access the model atmospheres, it may be because you are not currently using NDF data format (see command USENDF). Most of the data in SPECDAT is only available in NDF format. Of course, your node may not have the SPECDAT database installed, in which case complain to your node manager.

ATLIST *(no parameters)*

Lists the T(eff) and Log(g) values used to specify the Kurucz solar abundance model atmosphere fluxes that are accessible to ATLASRD.

ATNORM *[mode]*

Normalises model atmosphere fluxes stored in the 'current' arrays to the cursor position. The angular diameter implied by the normalising constant is printed at the terminal. The normalised fluxes are left in the 'current' arrays, and are plotted if mode=1 (the default). No plot is produced if mode=0.

If the cursor Y value is negative, the plot is assumed to be of X v Log10(FLUX). In this case, the Y values plotted (and left in the 'current' arrays) are logs of the normalised model atmosphere fluxes; however, the unnormalised atmosphere data in the 'current' arrays MUST be linear in Y to start off with (*i.e.* in the form resulting from an ATLASRD). ATNORM can also be used to normalise black-body fluxes generated with BBODY.

BBODY *temp*

Calculates black-body fluxes [pi times B(nu)] at the x values of the grid in the 'current' arrays, overwriting the y values therein. (GRID can be used to initialise appropriate x values.) The fluxes can be normalised to observed data using the ATNORM command. The unit of 'temp' is Kelvin, the 'x' values are assumed to be in Angstroms, and B(nu) is in erg/cm-2/s/A.

BEEP *(no parameters)*

Turns on the BEEP following NOBEEP (also tests your terminal's beeper).

BIN *X1 DX*

Bins the contents of the 'current' arrays, which are expected to be in monotonically increasing X order. X1 is the start wavelength of the input data, and DX the X range over which binning is to take place; thus the first X value in the output data set will normally be approximately:

```
X1 + 0.5*DX.
```

If you choose a value for DX which is less than the typical separation of the input X points you will probably get an unsatisfactory result. The binned data, which are the unweighted averages of the input X and Y values in each bin, are left in the 'current' arrays.

BOX *(no parameters)*

Automatic clearing of the plotting surface between plots. This is the default option on starting up. The inverse function is NB.

CDRAW *[filename or mode]*

Allows you to draw a 'continuum' using the cursor; the input X values must be in increasing order (CDRAW is terminated with an 'X(N+1).LE.X(N)' type test). The result is stored in the 'current' arrays (so you should first PUSH your spectrum so as not to lose it); subsequent rectification of data can be carried out using ADIV. (See also CREGD, CREGS and PF).

If MODE=0 (the default value), the data stored are just those input with the cursor, giving a 'join-the-dots' spectrum. If MODE=1, a spline fit to the data points is carried out, using the

subroutine INTEP described by Hill (Publ DAO). A smooth curve (which is supposed to be like one you might draw by hand through all the cursor points) is calculated on the grid of 'x' points of the arrays in the top (*i.e.* numerically largest) stack entry.

It is possible to input data to this routine from a file, rather than at the terminal. The data are expected to be in the form (NDF or native DIPSO) produced by the `WRITE` command, and MODE is assumed to be 1 (MODE=0 would have the same effect as a simple `READ`, and is therefore redundant). Thus, using this option makes `CDRAW` act essentially as a straightforward spline interpolation routine.

CLEAR ***(no parameters)***

Clears the text surface. (Simply a PRINT *, ' ' loop; it will not, therefore, work on a 4010-type device, for which you should use `ERASE`).

CLRBRK ***(no parameters)***

Clears all breaks (*i.e.* gaps in the data) from the current arrays. (The end-of-data is preserved, internally, as a break point, however.)

COMMANDS ***[clist/-word] [clist] [clist]***

If no parameters are supplied, a listing of all available commands is given. If a minus sign is given as the only parameter, a one line description of each command is also displayed. If the minus sign is followed by a word, then only those commands which contain the given word in their descriptions (case insensitive) are displayed. As an example:

```
> COMMANDS -CONTINUUM
```

would display all commands which contain the string "continuum" in their descriptions.

Alternatively, commands for display can be selected by their function. All commands are grouped into one or more classes which are identified by single lower-case letters as follows:

```
a - Analysis,measurement.
c - Setting of control variables, etc.
d - Modelling.
e - Data editing,selection,rejection.
f - Fitting.
g - Graphics.
h - Help information.
i - Reading, writing, moving and deleting data sets (including
    disk I/O).
l - Filtering.
m - Manipulation (arithmetic,calibration,corrections,etc).
q - Inquiry (control variables, etc).
r - Re-gridding.
t - Title manipulation.
y - System control tasks.
```

Each parameter must be a word made up from a selection of these class identifiers. Commands must belong to all the classes in at least one of these words to be displayed. Classes can be negated (i.e. explicitly excluded) by specifying an upper case identifier (i.e. "A" means "not in class "a""). As an example:

```
> COMMANDS gC t+
```

would list all commands which are to do with graphics but which are not control commands, and also all commands which are to do with title manipulation.

Note, if a file called `$LDIPSODIR/comand.lis` exists then the contents of the file are displayed without processing (all parameters are ignored). It is recommended that such files be re-formatted to use the new system (see section8.4).

CRASH ***n***

Performs various illegal instructions with the object of trying to crash the program! It can be used to test the behaviour of the signal/condition handler (see command `HANDLER`). The following values of 'n' causes the corresponding condition:

0 - Divide by zero (an example of a floating point exception).

1 - Access violation.

2 - Overflow.

3 - Underflow.

CREGD ***[h]***

'Continuum REGion Display': plots the regions selected for 'continuum' fitting using `CREGS`.

The continuum windows are indicated by horizontal bars, which are drawn a fraction 'h' of the distance from the bottom to the top of the plot ($0<h<1$, default 0.8).

CREGL ***(no parameters)***

'Continuum REGion List': lists the current continuum windows selected with `CREGS`.

CREGS ***[X1 X2 X3 X4 X5 X6 ... X49 X50]***

'Continuum REGion Select': select 'continuum' regions. These regions can be input as a parameter list; if no values are provided, the cursor is activated for interactive selection of continuum windows. When using the cursor, the input 'X' values *must* be in increasing order.

The selected regions can be checked using `CREGD`. This command will normally be used in conjunction with `PF`. (See also `CDRAW`).

CROT ***(no parameters)***

Implements automatic rotation of colours (when used with appropriate hardware). Each plot begins with the colour specified by the last call to `CSET` (or colour 1, if `CSET` hasn't been called).

To cancel, use `NCROT`.

CSET ***n***

Set colour for Ikon plotting; n = 1-12 (1 = white). (See also `CROT`).

CXR ***(no parameters)***

'Cursor X range': define the X range for plotting using the cursor.

CXYR ***(no parameters)***

'Cursor X & Y range': define the X and Y ranges for plotting using the cursor.

CYR ***(no parameters)***

'Cursor Y range': define the Y range for plotting using the cursor.

DEL ***n1 [n2 n3 n4 ..... n48 n49 n50]***

Deletes STACK entries. At least 1, and up to 50, stack entries may be deleted; after deletion, the remaining STACK entries are renumbered in sequence. Ranges of entries can be specified using the "-" operator; *e.g.* `DEL` 2 4-6 8 will result in the deletion of entries 2, 4, 5, 6 and 8. `DEL` 1-50 will clear the stack.

You are recommended to develop the habit of typing `SL` immediately after `DEL`, to check what has happened.

DEV ***workstation***

Opens a GKS workstation (plotting device). If the argument isn't provided then a table of legitimate workstations is listed, followed by a prompt. A full list of available workstations is given in SUN/83.

DRED ***E(B-V) [R MODE]***

Dereddens data stored in the 'current' arrays. Negative points are multiplied by -1, dereddened, and the dereddened values again multiplied by -1.

The default value of R (=A(V)/E(B-V)) is 3.1.

The default value of MODE is 0, which results in a 'standard' Galactic-type law being used (Seaton 1979 for the UV, Howarth 1983 for optical-IR); any non-zero value results in an LMC-type law being used (Howarth 1983).

DRLINE ***x1 y1 x2 y2***

Draws a poly line between two points on the current graph.

ECHO ***[mode]***

Controls echoing, at the terminal, of commands (and comments) read in from command files. If mode=-1 (the default) there is no echoing, but you get a "Command sequence completed" message when the macro file is closed; if mode=0 this message is suppressed. Mode=1 results in commands being echoed; mode=2 echoes comment lines (*i.e.* those beginning with "!" or "*"); and mode=3 echoes commands and comments. All non-zero modes give a "Command sequence completedi" on completion.

ELF ***(no parameters)***

This is *not a DIPSO command*, but a suite of programs accessed for Emission Line Fitting.

The available DIPSO/ELF commands are:

```
ELFINP   ELFOPT   ELFNEWC
ELFPUSH  ELFLFIX
ELFPUSHC ELFPOPC  ELFDELC ELFCSL ELFVUC
ELFSAVEC ELFRESTC ELFWRC
ELFPIN   ELFPL
```

The most important 'core' commands are:

```
ELFINP ELFOPT ELFPUSH
```

which are sufficient to give you a fit to some data.

ELFDELC ***[n1 n2 n3 n4 n5 n6 n7 n8 n9 n10]***

"ELF DELete Coefficients". Deletes entries from the stack of fit coefficients. Remaining coefficient stack entries are renumbered, so use of `ELFCSL` immediately following `ELFDELC` is recommended.

ELFNEWC ***(no parameters)***

"ELF NEW Coefficients". Clears the current array of fit coefficients (N.B. *Not* the DIPSO current arrays!). The same result can be obtained using `CLEAR` inside the `ELFINP` editor.

ELFINP ***[batch]***

"ELF INPut". Allows you to Specify starting values and constraints prior to optimising the line fits with ELFOPT. The "batch" parameter controls what happens if the ELFINP command is invoked from within a command file. If "batch" is zero (the default), the user is prompted for the strings specifying the ELF constraints, starting values, etc (you are alerted to this by a change of prompt). If "batch" is non-zero, the strings are read from the script file. In either case, the last string read should be "QELF".

The ELF command language allows starting values and constraints to be entered prior to optimising line fits. The language is similar to FORTRAN in the logical construction of commands; the following operators are recognised:

```
: = + - / *
```

The last four have their conventional arithmetic meanings.

Five variables are recognised:

```
C W I P D
```

These refer to the line centre position, line width (FWHM), peak intensity, profile type, and degree of background polynomial. (If no value is specified for the profile type, it is assumed to be a Gaussian; *i.e.* P=1). To refer to a particular line, the variable must be followed IMMEDIATELY by the appropriate index (*i.e.* C1, W5, I2 *etc*). Numerical constants can be input as integer or decimal numbers, but exponents are not accepted. Blanks are permitted, but not within variable names or numerical constants.

The ":" operator: Used to specify a starting value for a variable quantity, *e.g.*:

```
C1:5000
w1: 2.5
```

Starting values for peak fluxes are not required.

The "=" operator: Used to set a fixed quantity, or relationship between two quantities; for example,

```
C2=5000
w2=w1
I3 = 2.486 * I2
C3 = c1 - 21.6
p1 = 6
d=0
```

The command W2=W1 constrains the width of line 2 to be the same as that of line 1; P1=6 defines the profile type — in this case, to the first 'numerical' profile; D=0 fixes the background polynomial to have degree zero. (Note that the '=' operator is the only one allowed with the D and P variables.) If D is undefined no background polynomial is incorporated. (Note that least-squares polynomials can be fitted to data by defining no lines and an appropriate value of D. For reasons of numerical stability, the polynomial coefficients are computed using an x scale centred on the mean x value of the data set; you are informed of the value of this offset zero-point.) Other examples should be self-explanatory.

The "+" and "-" operators: These may ONLY be used with the variables C and W.

The "*" and "/" operators: These may *only* be used with the variable I.

The other commands available are:

```
HELP - gives a summary of options
CLEAR - clears the current model specification
QELF - return to DIPSO input
L - list the fit specification
```

There is no command for deleting the specifications for a given line from the complete coefficient specification. If you need to 'remove' a line, it is necessary to define (using the = operator) the line intensity to zero, and the line width and position to suitable arbitrary values.

ELFOPT *[prmpt]*

"ELF OPTimisation". Initiates optimisation of fit coefficients (in the sense of minimising the sum of the squares of the deviations of the fit from the spectrum data in the DIPSO 'current' arrays). In the case of an analytical profile fit, the data may have velocity or wavelength as the unit of the X axis. Provided WORV is set correctly, the fitted flux will be in sensible units (ie those of the Y axis). In the case of a 'numerical' profile, spectrum data *must* be in velocity units. Note that line width is *not* permitted as a free parameter when fitting 'numerical' profiles.

`ELFOPT` may not be fast on a busy machine if there are many free parameters and/or datum points. Also, for complex fits, `ELFOPT` may not converge on the correct solution. You may therefore want to monitor the progress of a fit on an iteration by iteration basis. To do this, you should specify the optional parameter "prmpt" to have a value of 1 or greater; you will then be prompted after the first (and, optionally, subsequent) iteration to give you the opportunity to exit `ELFOPT` cleanly.

The output from a completed fit consists of the optimised parameters and their errors (calculated in the linear approximation, from the error matrix). The results may be stored for later inspection, or output, using `ELFPUSHC`.

ELFPUSHC *(no parameters)*

"ELFPUSH Coefficients". Pushes the coefficients of the current ELF fit onto the stack of fit coefficients. These results may be inspected, printed out, or recovered to the status of current fit coefficients using `ELFCSL`, `ELFVUC`, `ELFWRC`, and `ELFPOPC`.

ELFPOPC *n*

"ELF `POP` Coefficients". Pops a set of fit coefficients from the ELF fit coefficient stack into the 'current' coefficient arrays (where they can be modified with `ELFINP`).

ELFRESTC *[filename[.typ]]*

"ELF RESTore Coefficient stack". Restores an ELF fit coefficient stack previously saved using `ELFSAVEC`. The SAVEd stack of numerical profiles is restored *only* if there are *no* numerical profiles on the profile stack at the time of the restore. This is to avoid ambiguity in definition of profile indices. The conditions and features of the restore are the same as those for the DIPSO stack restore ('RESTORE'), except that the default filename and type are [ELFSAVE.ESTK].

ELFSAVEC *[filename[.typ]]*

"ELF SAVE Coefficient stack". Saves the entire fit coefficient stack as an unformatted file that can be subsequently recovered using ELFRESTC. The stack of stored numerical profiles (if any) is also saved. The default filename and type are [ELFSAVE.ESTK]. The data file created is *not* an NDF and may therefore not be readable on operating systems other than the one on which it was created.

ELFCSL ***(no parameters)***

"ELF Coefficient Stack List". Gives a summary of entries in the fit coefficient stack. Individual entries can be inspected in more detail using ELFVUC.

ELFVUC ***n***

"ELF View Coefficients". Gives a listing, at the terminal, of entry 'n' of the ELF fit coefficient stack. An overview of the stack can be obtained using ELFCSL.

ELFWRC *filename[.typ]*

"ELF WRite Coefficients". Writes the contents of the fit coefficient stack to a file (default type is .DAT). The information given includes the starting specifications for the fit, the results (if any), and the line fluxes in units corresponding to the data stored in the DIPSO stack.

ELFLFIX ***n***

"ELF Line FIX". Allows use of the cursor to define a 'fixed' (*i.e.* non-optimisable) line in the data. Its principal use is to (in effect) take out features in the far wings of lines that are being optimised; these features might otherwise adversely affect the final fit.

Typing ELFLFIX brings up the cursor; two hits are then required. The first locates the centre and peak flux of the feature. (These data are added to the 'current' fit coefficients in the same way as using '=' in ELFINP.) The second locates the half intensity point, on either side of the feature, to fix the half-width at half maximum (from which the FWHM follows). (Hitting the same point twice leaves the width undefined.)

ELFPIN ***n***

"ELF Profile INput". Transfers a numerical profile from DIPSO stack entry 'n' to storage in the profile stack. The data *must* have units of velocity along the X axis; spectra to which numerical profiles are fitted must also be in velocity space.

ELFPL ***(no parameters)***

"ELF Profile List". Lists the contents of the profile stack (by giving the title of the original DIPSO stack entry).

ELFPUSH *[n1 n2]*

"ELFPUSH fit". Pushes the result of an ELF fit onto the DIPSO stack, as a continuous (*i.e.* no breaks) spectrum. If no arguments are specified, the complete fit is pushed, into the next available DIPSO stack position. If 'n1' is specified, only the fit for line n1 is pushed. If 'n2' is also specified, the fits for lines n1 to n2 inclusive are pushed. If n1 is specified, the background polynomial (if any exists) is also pushed, before the line(s).

ENV *name*

The string value being used for the specified environment variable is displayed. An error is reported if the no value is available.

DIPSO uses environment variables to specify various directories (eg `DIPSODIR`), and also the prompt string (`DIPSOPROMPT`). Default values for these environment variables can be supplied on the DIPSO command line in a comma separated list of "name=value" pairs. These default values will be used if the corresponding environment variables are not defined.

ERASE *[n]*

Erases plotting zone "n" (*c.f* `TZONE`). This is done rather slowly unless n=0 (the default).

EW *[mode]*

Measures equivalent widths. The cursor is used to define two pairs of (X,Y)points, between which the equivalent width is measured with respect to a linear 'continuum'; if you need a more complex continuum, the data can be preprocessed using `CREGS` together with `PF`, or using `CDRAW`. To terminate the equivalent width measuring session, define X2<X1. Note that `EW` expects to measure spectra with monotonically increasing X values.

Errors on the measured equivalent widths are calculated using the prescriptions given by Howarth and Phillips (MNRAS 222, 809, 1986); these errors are likely to make most sense for interstellar line measurements. The assumed nature of the errors is controlled by the value of MODE; the default for MODE is 0. In this case, if the data have been processed with `PF` immediately before using `EW`, an estimate of the continuum signal-to-noise is available. This results in automatic generation of 'statistical' errors on the equivalent width measurements, under the (conservative) assumption that noise (rather than signal-to-noise) is constant. Such an assumption is likely to be reasonable for IUE data. Note that on termination of the `EW` command the 'statistical' error is LOST. This is a feature, not a bug, and is intended to stop you making mistakes through oversight. If, after terminating `EW`, you find that you want to do further measurements on the same data, then the sequence:

```
PUSH,PF 0,ADIV m,PM
```

(where 'm' is the STACK entry into which the data are PUSHed) will normally recover the error estimates. (The `CREGS` continuum regions are remembered, and `PF` 0 will fit a horizontal line to these regions in the already rectified data, recomputing the error estimate. The Y coordinate of this line should be 1.0).

If MODE is given a non-zero value, 'statistical' errors are calculated under the assumption of Poisson statistics. Such an assumption may be considered by some (though not necessarily the author of this document) to be appropriate to IPCS data. Since the uncertainties on individual points are not stored, but are calculated (on a square-root basis) during the `EW` measurement, it is essential that *unrectified* data are interrogated in this mode. It may, of course, prove convenient to overplot a 'continuum' to assist interpretation.

In addition to the 'statistical' (*i.e.* random) errors calculated by `EW`, allowance for systematic errors in setting the zero and continuum levels can be made, using the `EWERR` command.

WARNING: `EW` operates on the data stored in the 'current' arrays. Be careful not to make the mistake of plotting STACK data, and then trying to measure equivalent widths off of the data on the screen. The calculated value of `EW` is multiplied by the factor WORV.

EWERR ***ErrC Err0***

Provides the program with estimates of the *systematic* errors in continuum and zero-level placement (expressed as percentages of the continuum level). These errors are then incorporated into subsequent error analysis when determining equivalent widths with `EW`.

These systematic errors are assumed to propagate quadratically (see MNRAS 222, 809, 1986).

EXIT

Exits DIPSO, saving the stack in a file called EXIT.STK, or EXIT_STK.sdf.

EXPAND ***Factor [clist]***

Expands components of a plot by a factor "Factor" with respect to the default size on a given device. (Factor is absolute, not relative; *e.g.* the sequence `EXPAND` 2, `EXPAND` 3 gives 3x enlargement, not 6x.)

If no value is supplied for `clist`, then the supplied expansion factor applies to all the components of the plot listed below. If a value is supplied for `clist`, it must be a string containing some sub-set of the characters A, I, N, T, M and P. Each character refers to a different component of the plot as follows:

A - M(A)jor tick marks.

I - M(I)nor tick marks.

N - (N)umerical axis labels.

T - (T)extual axis labels.

M - (M)arkers.

P - Text created using the (P)WRITE command.

The expansion factor is only applied to those components of the plot for which the corresponding letters are included in the `clist` parameter. The expansion factor for all other components is left unchanged.

FILL ***(no parameters)***

Results in filled-in MARK symbols (*c.f* NOFILL).

FLUX ***[filename[.typ]]***

Measures fluxes with respect to (=above) a linear 'continuum' defined using pairs of cursor hits. More complex continua must be rectified out, using CDRAW, or CREGS together with PF. To end a FLUX measuring session, input X2<X1. Note that FLUX expects to measure spectra with monotonically increasing X values.

Fluxes are obtained by trapezoidal integration between X,Y points. Linear interpolation is used across 'breaks' in the data (and a warning given).

If a file name is supplied (the default file type is .DAT) then the x1, x2 and flux values are output to the file for future reference. The file remains open until (i) you exit from the program; (ii) a new file name is provided; or (iii) FLUX 0 is typed in. (The last option closes any file that is open [but does not open a file called 0.DAT].) This means that, for example, FLUX can be exited, the X and Y ranges changed and a new plot generated, then FLUX re-entered, and data will continue to be output to the previously opened file.

WARNING: FLUX operates on the data stored in the 'current' arrays. Be careful not to make the mistake of plotting STACK data, and then trying to measure fluxes off the data on the screen. The integrated flux is multiplied by the factor WORV.

FONT ***n***

Selects font quality. Permitted values of 'n' are 0 (hardware characters), 1 (SGS characters), and 2 (NCAR characters). These are in increasing order of elegance, and execution time! The resulting fonts are used for all text on the plot (*i.e.* apply to XLAB, YLAB, PWRITE, and TITLE when the title is plotted).

FONT 0 (the default) is not always elegant, and because of problems with text rotation some devices will write the Y axis label "upside down". If this really worries you you can get round it with YLAB (*e.g.* use YLAB xulF).

The elaborate FONT 2 style gives access to a wide range of special characters (Greek, italics, mathematical symbols, *etc.*) These cannot be reproduced in this document, but are given in full in SUN/90 (which documents the routines which DIPSO utilises). The following formats serve to illustrate the possibilities:

'PGU' gives uppercase Greek

'PGL' gives lowercase Greek

'B' gives subscripts ("B"elow)

'S' gives superscripts

'N' gives normal (ie not sub- or super-script)

'PRU' gives Roman characters

Thus to give the formula for the area of a circle as a title:

```
TITLE "Area = 'PGL'P'PRU'r'S'2"
```

Note that the single quotes surrounding the style definition strings are mandatory, as are the uppercase specifications. Note, too, that the correspondence between Greek and Roman characters is not usually as obvious as P="pi"; it is important to check a printed (not line-printer!) copy of SUN/90 for details.

An angstrom symbol (Å) is provided as a special character: '.A' (in 'PRU' style). Other complex characters can be constructed (see SUN/90); for example, a mass-loss rate symbol (M surmounted by a dot) is obtained using `"'PRU'M'H:-85V105PRU'.'H:85V-105'"`.

Because the codes for special characters are flagged by apostrophes, life gets complicated if you want to include a normal apostrophe in a tring. But not too complicated: you just give it twice. For example, if you want a title that says:

Howarth's Fudged Data

you must specify:

```
TITLE "Howarth''s Fudged Data"
```

to get it (in font 2).

FORMWR ***filename[.typ]***

Results in a formatted write of the contents of the 'current' arrays. The resulting data are in a form suitable for output on a line printer. The default file type ([.typ]) is '.DAT'.

FRAME ***xsize ysize [locator index]***

Defines a plotting area in absolute (device-independent) terms. The xsize and ysize parameters are in cm; the locator index follows the numeric keypad, i.e. 1=bottom left, 9=top right *etc.* The locator index defaults to 5 (centre of plotting zone). To return to (device-dependent) plotting zones use `TZONE`.

Subzones within the specified `FRAME` can be set using `FRZONE`.

FRZONE ***x1 x2 y1 y2***

Defines subzones within a plotting area specified by `FRAME`. The parameters x1 *etc.* are in the range 0-1, and define the fractional position of the subzone within the frame. An example of the use of `FRAME` and `FRZONE` can be found in `DIPSODIR:DEMO2.CMD` (`$DIPSODIR/demo2.cmd`.

FTFILTER ***n1 [f1 f2 n2]***

Filters high-frequency components (which you may identify with noise) from the real and imaginary parts of a Fourier Transform (see FTRANS). FTFILTER expects the real part of the FT to be in stack entry “n1", and the imaginary part to be in entry “n2" (which defaults to n1+1).

FTFILTER operates by constructing an “exponent-squared edge function", and multiplying the FT by that filter (see Bracewell, “The Fourier Transform and its Applications", for details). The filter has a value of unity up to frequency f1, then drops like exp(D[nu]**2), where D[nu] is the change in frequency. The e-folding scale is determined by specifying f2, the frequency at which the filter drops to a value of 0.01.

The filtered real and imaginary parts are pushed onto the stack, together with the filter used.

The frequencies f1 and f2 will depend critically on your application; defaults are provided merely to provide an illustration for first-time users. These defaults are (Nu1+Nu2)/2 for f1, and MIN(Nu2,1.1*f1) for f2, where Nu1 and Nu2 are the first and last frequencies in the data set to be filtered.

FTINV ***n1 [n2]***

Computes the inverse fourier transform from the real and imaginary parts of the FT, where the real part is in stack entry n1, and the imaginary part is in stack entry n2 (which defaults to n1+1). The result is pushed onto the stack (and WORV=1 assumed).

FTRANS ***n [p]***

Computes the Fourier Transform of stack entry “n”, pushing the resulting real and imaginary parts onto the stack, together with the power spectrum. A fraction “p” of the data set is endmasked (at each end) with a cosine bell; “p” defaults to 0.05. For best results you should subtract a continuum (or at the very least the mean value) from the data, in order to maximise the effectiveness of the endmasking.

FTRANS does not use the Fast Fourier Transform algorithm. It does not, therefore, require the number of datum points to be an integer power of two (at the price of being slow for large data sets). However, it is necessary for the data to be uniformly sampled; if FTRANS determines that your data set does not contain regularly sampled X values it will automatically perform a 4-point Laguerre interpolation in order to simulate such data. For applications where the data sampling rate is very irregular, or where there are substantial gaps (*e.g.* in light-curves), it is recommended that PDGRAM (and PDGWINDOW) be used for Fourier analysis applications.

The X values in any data set must be monotonically increasing.

It is recommended that you familiarise yourself with (*e.g.*) Brault & White, A&A 13, 169 (1971) before using FTRANS.

GRID ***X1 X2 DX***

Creates a grid of uniform wavelength points, starting at Lambda(1)=MIN(X1,X2). The Y values are set to zero. This command can be used with AADD to remap data, or to set up a wavelength grid for (*e.g.*) NEBCONT. Data are left in the ‘current’ arrays.

GRIDSTYLE *mode*

Controls the design of plots.

- Mode 1 is the “ordinary" (4 sides and labels) design;
- Mode 2 results in a grid drawn over the plot (like course graph paper);
- Mode 3 gives just the bottom and left-hand axes;
- Mode 4 gives no box at all, and no labels;
- mode 5 gives bottom and left-hand axes but no labels.

HANDLER *level*

The DIPSO error handler makes the program uncrashable (in principle!). Setting level=0 results in the system error handler being invoked; fatal errors will kill the program. Any level greater than zero (such as the default value of 1) causes the program to issue a warning message when an error occurs (including control-C interupts). The current command is aborted and the user is returned to the main DIPSO command prompt.

To familiarise yourself with the condition handler you can type ‘CRASH’.

HC *N*

Calculates a theoretical (fully damped) profile for the Hydrogen Lyman-alpha line (1216 A), with a column density ‘N’. The profile is calculated at the X grid in the current arrays, with a continuum level of unity. If the parameter N is less than 30 it is assumed to represent log(N).

The most convenient way of using this function is to calculate a theoretical profile, use `ADIV` to divide observed data (in the STACK) by it, then plot the resulting data. The ‘best’ value of the interstellar H I column is that which gives the ‘flattest’ continuum around 1216. Of course, interpretation is your responsibility, and the intrinsic stellar Lyman-alpha profile should always be considered, together with the possibility that the interstellar line is not fully damped.

HELP *[string]*

If no value is supplied for ‘string’ then general help information for the DIPSO package is displayed. More verbose assistance on a particular command can be obtained by typing ‘`HELP` <command name>’. The information is displayed in one of two formats: plain text, or hypertext. The hypertext information is taken from SUN/50 and is displayed using a World-Wide-Web browser. By default the *Mosaic* browser is used, but this can be changed by assigning the required browser command to the environment variable `HTX_BROWSER` before entering DIPSO. For example, to use *Netscape* issue the following command before starting DIPSO:

```
% setenv HTX_BROWSER netscape
```

If a browser of the requested variety has already been fired-up prior to starting DIPSO, it is "hi-jacked" to show the requested help information. Otherwise, a new browser is fired-up.

If this command fails, saying that the "showme" command cannot be found, try defining the showme command explicitly using `USEHTX`.

In plain text mode, the information is taken from `$DIPSODIR/dipso.hlp` and `$DIPSODIR/help.lis`, and is displayed in the DIPSO command window.

By default, the plain text format is used, but this can be changed using the `USEHTX` command.

Typing an interrogative ("?" or "?<command name>") has the same effect as typing `HELP`.

HIST ***(no parameters)***

Plots to be done histogram-style (as opposed to `POLY` or `MARK`).

HPROT ***[mode]***

Rotates between Hist and Poly plots on a given diagram, starting with whichever style is currently in force (selected with `HIST` or `POLY` command); can be useful when comparing (*e.g.*) observations and models. If mode is 1 (default is 0) then the first data set is plotted in the current style (*i.e.* histogram if `HIST` is in force), and all subsequent data sets plotted on the given diagram will appear in the alternative style. The cycle can be re-initialised at any time with `HIST` or `POLY`.

Negated with `NHPROT`.

INTEGRATE ***(no parameters)***

Estimates the area under the current arrays using simple trapezoidal integration (and will therefore only work successfully for data sets with monotonically increasing X values).

INTERP ***n***

Applies Laguerre interpolation to data in the current arrays, to give a regularly sampled data set with the same number of points as the original. The parameter "n" controls the order of the interpolation.

`INTERP` does (n-1)th order interpolation, where 'n' *must* be an even number; thus only odd orders greater than zero are handled. For example, n=2 gives 1st order (*i.e.* linear) interpolation; n=4 gives 3rd order (i.e. quartic) interpolation. (INTERP cannot supply quadratic interpolation, since this would require n=3 and odd values of 'n' are not allowed.)

The data must have monotonically increasing X values.

Note that spline interpolation can also be executed, using the `CDRAW` command.

ISATM ***[wavelength]***

Supplies atomic data for use with `ISCALC` and `ISCOG`. If a wavelength (in Angstroms) is provided, then DIPSO searches for a file called ATOMIC.DAT in the current directory; if it fails, it looks for one in a directory with the environment variable OWNERDIR; and if it again fails, it opens `$DIPSODIR/ATOMIC.DAT`. Once a file has successfully been opened, DIPSO searches for a line

whose wavelength falls within 0.1A of the argument wavelength to ISATM. If it finds it, it uploads the associated atomic data.

The file $DIPSODIR/ATOMIC.DAT uses data based on MNRAS 222, 809 (1986) and references therein, and provides a model for the required format if you want to set up your own file. "Comment cards" prefixed by "*" or "!" are permitted.

If a wavelength is not provided, or if ISATM fails to find a data set corresponding to a specified wavelength, then a "data edit" mode is entered. This allows you to change or input atomic data, or to carry out further searches through an ATOMIC.DAT file; typing H in this mode gives more information.

(Incidentally, in case you're wondering, ATOMIC.DAT requires statistical weights in order to permit calculation of damping constants.)

ISCALC ***(no parameters)***

Calculates a theoretical absorption profile for an interstellar cloud (or other plane-parallel slab of absorbing material with negligible forward scattering and a Gaussian line-of-sight velocity distribution for the absorbers), and pushes the result onto the stack.

Before this command can successfully be invoked, the atomic data must already have been loaded (with ISATM), a cloud model defined (with ISINP), and a set of cloud 'options' defined (with ISOPT).

ISCOG ***(no parameters)***

Calculates a Curve-Of-Growth for a cloud model defined with ISINP, using atomic data input via ISATM. The COG is pushed onto the stack, with X values (N.f.lambda) in units of (cm-2.dimensionless.Angstroms) and Y values (W[lambda]/lambda) assuming equivalent width and wavelength to be measured in the same unit.

ISINP ***(no parameters)***

Invokes a cloud "editor", which permits an interstellar cloud model to be defined prior to calculating theoretical profiles with ISCALC. Up to 18 clouds are permitted; each is specified by a velocity dispersion parameter, "b" (see, *e.g.*, MNRAS 222, 809 [1986] for a definition), a central velocity, V, and a column density, N.

Typing HELP while in the cloud editor provides more information.

ISOPT ***[filename[.typ]]***

Loads a variety of options required before ISCALC can be successfully executed. These include specifications for (optional) convolution with an instrumental resolution function, blending with other lines, *etc.* ISOPT invokes an option "editor"; typing H in this edit mode gives more information. Alternatively, if a filename is specified on the command line, an attempt is made to read options from that file. Such a file should contain information matching the ISOPT editor input; for example, if the file contains:

```
V1=-100
V2=+100
```

line profiles will be calculated over (at least) the range -100 km/s to +100 km/s; default values for all other options will be assumed. (The default file name is ISOPT, and the default extension is .DAT.)

IUECOR ***camera year day [aperture]***

Applies the 'aging' corrections for the IUE cameras described by Bohlin and Grillmair (*Ap. J. Sup.,* **66,** 209, 1988; SWP) and by Clavel *et al.* (ESA IUE Newsletter No. 26, p. 65, 1986; LWR). The 'camera' parameter is 2 for LWR and 3 for SWP; the 'day' parameter is day number in the year. The 'aperture' parameter is relevant only to SWP data, and is 1 for trailed spectra, 2 for small aperture spectra, and 3 (the default) for large aperture point source spectra. IUECOR works on data in the current arrays, replacing fluxes therein by their corrected values.

LABON ***(no parameters)***

Turns axis labelling back on (after use of NLAB).

LOGAXX ***t/f***

If set 'True', the X axis will be plotted on a log10 scale.

LOGAXY ***t/f***

If set 'True', the Y axis will be plotted on a log10 scale.

LOGX ***(no parameters)***

Replaces the X values in the current arrays by their base 10 logarithms.

LOGY ***(no parameters)***

Replaces Y values in the current arrays by their base 10 logarithms.

LWEIGHT ***weight***

Alters the weight (*i.e.* 'heaviness') of all plotted lines, on devices which support this feature. The weight must be 1-5 (initial setting is 1). See also TWEIGHT.

MARK ***(no parameters)***

Plots to be done using symbols, as opposed to HIST or POLY. The symbols may be 'designed' using MSET, and expanded and rotated using EXPAND and ANGLE.

MEAN ***(no parameters)***

Calculates the mean and standard deviation of the Y data stored in the 'current' arrays.

MERGE ***S1 S2 WT1 WT2 [MODE]***

Merges STACK entries S1 and S2, with weights WT1 and WT2 respectively. This function is an alternative to the AADD and YMULT commands, differing in the respect that if there is a gap in one (but not both) data sets being manipulated, then the MERGEd data set will not have a gap. The data sets in the STACK are both required to have monotonically increasing X values.

If there is any region of overlap between the two data sets, then the Y values of the data set associated with the larger value of X(1) are mapped onto the X grid of the other data set. This remapping is done using a triangular filter, so that some slight smoothing can result if the sampling rates in the two data sets are very different. Remapping is carried out ONLY in any overlap region.

This function is intended for merging data sets with (normally) overlapping, but not necessarily identical, X ranges. In particular, it can be used for merging IUE SWP and LWR flux calibrated spectra, and UV and optical data (*e.g.* UBV fluxes; see UBVRD). Because remapping is done linearly, it is also suitable for averaging similar spectra.

If MODE=zero (the default value) and the data look like IUE SWP and LWR spectra (on the grounds of the X ranges), a wavelength-dependent weighting is used which reflects the inverse sensitivity function of IUE. If you ARE dealing with IUE data, then inputting the exposure times as the WTs and using MODE=0 will result in (essentially) a MERGE at the 'FN' level, which is probably more correct than one at the 'absolute flux' level. If MODE is positive, the additional IUE weighting is switched off. (Note that if you are NOT working with IUE data MODE=0 and MODE=1 will give the same result).

If MODE=-1, a straight addition of data in the overlap region is carried out. This option may be useful for (eg) combining emission line models with a zero background level. If MODE=-2 (or less), the two data sets are multiplied in the overlap region. This option may be useful for (*e.g.*) combining absorption line models with a background level of unity. If MODE is negative, 'dummy' values of Wt1 and Wt2 MUST be supplied.

You should only merge data which you expect to have similar Y values over any range of overlap. Merging dissimilar data sets (*e.g.* SWP and LWR IUE spectra which have not been flux calibrated) will give unsatisfactory results.

MONGOWR ***filename [badval]***

Writes a file which can subsequently be read into MONGO. It is possible to make MONGO leave gaps in plots, corresponding to datum points with a notifiable 'bad' y value; the MONGOWR command inserts a 'bad' point with y value 'badval' (default 0) into breaks in the data set to facilitate use of this option.

Most of the more important MONGO functions are reproduced in DIPSO - you shouldn't have to use this command!

MROT ***(no parameters)***

Implements automatic rotation of plotting symbols. Each plot begins with the symbol type specified by the last use of MARK (or symbol 1, if MARK hasn't been called).

NOT IMPLEMENTED AT PRESENT.

Turned off with NMROT.

MSET ***style nvert***

Selects the MARK plotting symbol. 'NVERT' is the number of vertices the symbol has; 'STYLE' takes on values 1-4. Style=1 is a polygon, Style=2 a 'star' design, style=3 is an asterix, Style=4 gives an arrow symbol, for plotting lower or (with ANGLE) upper limits.

NB ***(no parameters)***

'No Box': switches off automatic clearing off the plotting frame between plots. (Inverse function is BOX). On a device or zone change, the 'box' is switched 'on' for the first plot, regardless of any NB call.

NCROT ***(no parameters)***

Stops automatic rotation of the colour table when Ikon plotting and resets the colour index to 1 (white). (See CROT and CSET).

NEBCONT ***filename[.typ] [mode1 mode2 mode3]***

Calculates a theoretical nebular recombination continuum. The requisite data are comparatively numerous, and so are accessed from a file. The contents of the file must be as follows:

```
Line 1: Te(1-4)
Line 2: Ne
Line 3: Log10 H(beta) flux [and Log10 He(1640) flux]
Line 4: C
Line 5: A[He(1+)] A[He(2+)]
Line 6: A[N(1+)] A[N(2+)] A[N(3+)] A[N(4+)]
Line 7: A[C(1+)] A[C(2+)] A[C(3+)] A[C(4+)]
Line 8: A[O(1+)] A[O(2+)] A[O(3+)] A[O(4+)]
Line 9: A[Ne(1+)] A[Ne(2+)] A[Ne(3+)] A[Ne(4+)]
```

where:

Te(i) is the electron temperature appropriate to ions of charge i+, in units of 10^4K;

Ne is the electron density in cm-3;

H(beta) flux is the observed value, in cgs units; if the flux is unknown, enter a value of 0.0 and follow it (on the same line) with the log of the He II (1640) flux, in the same units;

C is the logarithmic extinction at H(beta) [C=1.44*E(B-V) for a standard extinction law];

A(X) is the ionic abundance, in the form 1000*N(X)/N(H+).

A specimen file is given in $DIPSODIR/NEBCONT.DAT. If NEBCONT fails to find the file in the current directory, it tries to look in a directory assigned the environment variable OWNERDIR (which you can define in your .login script or your LOGIN.COM procedure).

All data in the file are reproduced at the terminal. By default, (mode1=mode2=mode3=0), new values are prompted for (simply hit 'return' to get the file values). By setting mode1=1, abundances are listed but not prompted for (fluxes, C, temperatures and densities still prompted). Mode2=1 does the same for the H(beta) flux and 'C'; and mode3=1, the same for temperatures and density.

The results are calculated at the X co-ordinates of data in the 'current' arrays, and pushed onto the stack. If you are not modelling real data, `GRID` can be used to set up an X array.

The default file type is `.DAT`.

NECHO ***(no parameters)***

Has the same effect as `ECHO` -1.

NLAB ***[mode]***

Selectively turns off axis labelling (modes not yet implemented; `NLAB` currently turns off X and Y axis labels, and title).

NHPROT ***(no parameters)***

Stops rotating plot style between Hist and Poly (after `HPROT`).

NMROT ***(no parameters)***

Stops automatic rotation of plotting symbols and sets the `MARK` index to 1. (See `MROT` and `MARK`).

NOBEEP ***(no parameters)***

Stops the terminal beeping every time you make a boo-boo.

NOFILL ***(no parameters)***

Results in `MARK` symbols being plotted as open (unfilled) figures; *c.f.* `FILL`.

NTROT ***(no parameters)***

Stops automatic rotation of line attributes and resets the `TLINE` index to 1 (continuous lines). (See `TROT` and `TLINE`).

NROT ***(no parameters)***

Stops rotating everything, resetting the colour, `MARK` and `TLINE` indexes to 1 (i.e. has the same effect as NCROT,NMROT,NTROT).

NX ***(no parameters)***

Return to autoscaling of the X axis (after using `XR`, `XMIN`, `XMAX`, `CXR` and/or `CXYR`).

NXY ***(no parameters)***

Return to autoscaling of the X & Y axes (after using XR, YR, XMIN, XMAX, YMIN, YMAX, CXR, CYR, and/or CXYR).

NY ***(no parameters)***

Return to autoscaling of the Y axis (after using YR, CYR and/or CXYR).

PAUSE ***(no parameters)***

Gives a BEEP (unless you've specified NOBEEP), and causes nothing to happen until you hit the return key. (You might find a use for this command in command files.)

PF ***n***

'Poly fit': fits a polynomial of degree 'n' through data points in the HIGHEST (*i.e.* numerically largest) STACK entry, using continuum windows defined using CREGS. PF expects these data to be in order of increasing X value. Y values, obtained from the parameters of the 'best fit', are calculated at the grid of X points of the data in the highest STACK entry, between the first and last 'continuum window' X values. The resulting curve is left in the 'current' arrays (overwriting anything there previously).

Because the 'continuum window' data are stored internally, it is possible to carry out a sequence of trial fits in an attempt to obtain a satisfactory polynomial representation of the continuum. Thus a typical sequence of commands to rectify data might be as follows:

```
POP n, PUSH       ! Put the  data  of  interest  into  the
                  ! 'current'  arrays and onto the top  of
                  ! the stack.
XR x1 x2,PM       ! Set the X range and plot the data.
CREGS,CREGD       ! Select & display  continuum windows.
PF 1              ! Fit  a  straight line to the  'window'
                  ! data.
NB,PM             ! Plot the fit through the data.
PF 3              ! PF 1 unsatisfactory; try again.
PM                ! Plot the fit; it looks O.K.
ADIV m            ! Divide data in the top of the STACK by
                  ! by the polynomial approximation to the
                  ! continuum.
TITLE new title   ! Change the title appropriately.
BOX,PM            ! Plot the rectified data.
PUSH              ! Save the rectified data for later use.
POP j,PUSH        ! Prepare the next data set of interest.
PM,NB             ! Plot the data.
PF 3,PM           ! Fit & plot a polynomial; there is  no
                  ! need  to  redefine  continuum  windows
                  ! unless you want to change them.
```

(Then ADIV, TITLE, and so on).

Note that a slightly different approach to polynomial fitting is possible using ELFINP and ELFOPT (which can give you the coefficients of a least-squares fit).

PDGPEAK ***(no parameters)***

Locates the peak of the data set in the current arrays (notionally, but not necessarily, produced with PDGRAM). PDGPEAK does this by locating the largest Y values, then fitting a parabola to the point and the adjacent ones on either side. The peak and central values listed are those calculated from this parabola.

PDGRAM ***[fl fh df p]***

Replaces the data in the current arrays (which should have monotonically increasing X values) with their "unevenly spaced data periodogram", as defined by Scargle (ApJ 263, 835, 1982).

The parameter fl is the lowest frequency at which the periodogram is to be evaluated, and defaults to zero; fh, the highest frequency, defaults to 0.5*(n-1)/(x[n]-x[1]), where there are "n" points in the current arrays; df, the frequency interval, defaults to 1.0/(x[n]-x[1]); and p, the proportion of the data set endmasked at each end (using a cosine bell), defaults to 0.05.

N.B. For large data sets the default frequency parameters may lead to very large numbers of points in the periodogram, which will accordingly take a substantial time to evaluate.

The periodogram has X units which are the inverse of the original X units (ie will normally be in units of spatial or temporal frequency).

PDGWINDOW ***[fl fh df]***

Replaces the data in the current arrays (which should have monotonically increasing X values) with their window function, as defined by Scargle (ApJ 263, 835, 1982). The parameters fl, fh, and df have the same meanings and default values as for the PDGRAM function, and the caution given for that function regarding lengthy evaluation times applies also to PDGWINDOW.

One would normally calculate a data set's window function in addition to its periodogram (PDGRAM) to ensure that any peaks in the latter are not an artefact of the sampling frequency.

PLOTINV ***(no parameters)***

Invert the Y axis on plotting (*i.e.* the 'bottom' Y value becomes the 'top' value, and vice versa).

PLOTREV ***(no parameters)***

Reverse the X axis (ie the X value at the left-hand edge of the plot becomes that at the right-hand edge, and vice versa).

PM ***[n1 n2 n3 ..... n48 n49 n50]***

Plot data ('PM' comes about 'for historical reasons'). If you haven't used the PPROMPT command, then PM without any arguments will plot the data stored in the 'current' arrays; otherwise up to 50 STACK entries can be plotted. If you have set PPROMPT to TRUE, then if you didn't provide any arguments for PM on the command line DIPSO will prompt you for some when it comes to do the plot. In order to be able to plot both 'current' and 'STACK' data, the 'current' arrays are awarded the 'honorary' STACK entry number 0 for this command *only*. Ranges of stack

entries may be specified using the "-" operator; *e.g* PM 1 3-5 will result in entries 1, 3, 4 and 5 being plotted.

Even if the BOX is switched 'on', all the data associated with a single PM command are plotted in a single frame; *i.e.* a command sequence like:

```
BOX,PM 1 2 0 4
```

will have a different result from:

```
BOX,PM 1,PM 2,PM 0,PM 4
```

WARNINGS:

- If you try to plot more than 50 spectra, numbers 51 et seq will *not* be plotted; moreover *no* error message will be given. You will have to attempt to plot the 'current' arrays and the entire contents of a completely full STACK (or multiply plot the same data set) for this problem to arise.
- If autoscaling is in effect, the plotting frame is autoscaled to the X,Y data in the *first* stack entry specified. Thus a command like:

  ```
  PM 1 2 3 4 5
  ```

  may produce a different plot to:

  ```
  PM 5 4 3 2 1
  ```

POLY ***(no parameters)***

Plotting to be done 'join-the-dots' style (as opposed to MARK or HIST).

POP ***n***

Pop STACK entry 'n' into the 'current' arrays.

PPROMPT ***switch***

PPROMPT governs the default action of PM. The value of the 'switch' argument is T(rue) or F(alse), and on startup is set to F. If you set it to T, then the PM command will prompt for an argument list if none is provided on the command line. This could be useful if, for example, you want to include PM in command files where you may not know in advance which stack entries you want to plot. If the switch is set to F, then typing PM without any argument results in the data in the current arrays being plotted.

PS ***X1 X2 [DX] [n1 n2 n3...n50]***

'Plot Spectrum': this command, which acts on the contents of the 'current' arrays, is intended specifically for plotting long stretches of a single spectrum (*e.g.* high resolution IUE data) in sequential frames. The parameters are:

X1: the X value at the left-hand edge of the first frame;

X2: the X value at the right-hand edge of the first frame;

DX: the amount by which X1 and X2 are incremented in successive frames. DX defaults to (X2-X1).

Plots are alternated in zones 5 and 6. After each frame is plotted, you are prompted as to whether or not you want to continue plotting.

WARNING: On completion of this command the X range and plotting zone in force will be those used for the last frame; you will probably want to change them.

PUSH ***(no parameters)***

'Pushes' the contents of the 'current' arrays onto the top of the STACK.

PWRITE ***x y i string***

Writes a character string 'string' at co-ordinates x,y. If `FONT` 2 is active, the "locator index", 'i', determines the location of the string with respect to the x,y coords: if i=1, the string is written to the lower left (*i.e.* the top right corner of an imaginary box containing the string ends at x,y); if i=2 the 'box' is horizontally centred on, and vertically below, x,y; i=3 has the top left corner of the 'box' at x,y; i=4,5,6 correspond to the 'box' vertically centred and to the left, centre, right of x,y, respectively; i=7,8,9 are similar, but above x,y. These locator indexes are chosen so that the numeric keypad available on most keyboards acts as a mnemonic.

If the string contains commas, it *must* be enclosed in double quotes (`"`). It is recommended that you develop the habit of using such quotes in any case.

The `ANGLE`, `EXPAND`, and `FONT` commands can be used to modify the appearance of the string on output.

QAREA ***Zn***

Reports the (relative) sizes of the grid and graph windows for zone "Zn", together with the absolute area of the plotting surface (if available). See the `TPORT` command for details.

QSM ***Sigma***

Applies a 'quick' Gaussian smoothing (FWHM of filter = 2.354*Sigma) to Y values in the 'current' arrays. (Defines gaussian weights at a grid of delta(x) values, then linearly interpolates when smoothing). Much faster, and scarcely less accurate, than `SM`.

QUIT *(no parameters)*

Quit DIPSO, without saving the stack. Can be abbreviated to "Q".

RDCAT *file xcol ycol*

Reads the values from two specified catalogue columns in the the current arrays. The file containing the catalogue is given by "file". This must be a full path name (no shell meta-characters such as `~`, `$`, etc can be used here). The file can be a FITS table (binary or formatted), or an STL table (see SUN/190). The names of the columns holding the X and Y values are given by "xcol" and "ycol". A full list of available column names is displayed if an unknown column name is supplied.

READ *file [sys] [unit]*

Reads a disk file which has been written with `WRITE`. This and `RESTORE` are the only input modes that preserve ALL the information associated with a DIPSO data set, and are the recommended options. The disk data can be in either NDF or "native DIPSO" format, depending on the flag established by the `USENDF` command. When specifying an NDF, do not include any file type.

If the specified NDF was not created by DIPSO, then an attempt will be made to create appropriate X values based on the WCS component in the NDF (or the FITS header cards in the FITS extension if there is no WCS component). The second and third command parameters, "sys" and "unit", may be used to indicate the spectral system and units in use within the NDF (these are only used if the system and units are not clearly specified in the NDF). The system value supplied must be a standard FITS-WCS value such as "FREQ", "WAVE", "VRAD", "VOPT", etc. Likewise, the unit should be a standard FITS-WCS such as "m", "nm", "Angstrom", "Hz", "GHz", "m/s", "km/s", etc. The spectral axis values within the NDF will be converted automatically from their original system to either wavelength in Angstroms or optical velocity in km/s for storage in the DIPSO X array. The original system and unit adopted for the NDF during this conversion are reported. If the user does not specify a system and/or unit, then a default of "wavelength in metres" will be adopted unless the contents of the NDF suggest some other default.

RECA *[/all] [string or integer]*

Recalls a previous command line or prompt response, putting it into the current keyboard input buffer for acceptance (by pressing return) or further editing. The `RECA` statement can be issued *when-ever* dipso prompts the user for input. If no parameters or qualifiers are given, the string which the user last typed in is recalled. If a non-numeric parameter value is given, the most recent string to be given by the user which starts with the supplied string parameter is recalled. If a numeric parameter is given, the corresponding string is recalled where "1" is the last (i.e. previous) string, "2" is the last-but-one, etc. If the `/all` qualifier is given (i.e. "`RECA/ALL`"), a list of the last 20 strings given by the user is displayed, and the user is then re-prompted.

RECALL *(no parameters)*

Lists a `RECORD`ed string at the terminal (as a reminder!)

RECORD *"string"*

Records a string of commands for subsequent REPLAY. The string must be enclose in double quotes; it can itself contain double quotes, but must then be delimited by pairs of double quotes. The string may not incorporate RECORD or REPLAY commands, but the syntax, and general validity, of the string is otherwise not checked until it is REPLAYed.

To cancel a current RECORDing, use a null string.

REPLAY *(no parameters)*

If REPLAY is included in a command line, then it is replaced in that command line by the contents of any RECORDed command string.

REPORTING *level*

This command can be used to reduce the number of error messages displayed if one of the NDF accessing commands fail. The command takes a single integer parameter. Level = 2 is the initial value and results in all error messages being delivered to the screen. Level = 1 replaces all errors deriving from a single failed command with a single (less detailed) error message. Level = 0 suppresses all error reports.

RESTORE *[file name or NDF name]*

Restores STACK data previously dumped using SAVE. RESTORE does not require the current stack to be empty; however, if it is not then retrieval may not be complete, depending on the available stack space. (Notification is given of incomplete retrieval, of course). The data can be in either NDF or "native DIPSO" format, depending on the flag established by the USENDF command. When specifying an NDF, do not include any file type. The default NDF name is 'SAVE_STK'. If not using NDFs, the default file name is 'SAVE.STK'.

RETITLE *n string*

Changes the title of stack entry "n". The string (which may be null) must be enclosed in double quotes if it contains commas. (The use of double quotes is in any case recommended.) Special characters may be incorporated (see the FONT command).

RXR *X1 X2*

Restrict the X range of data in the 'current' arrays (*i.e.* throw away data outside the range X1 to X2). Only works properly on data sets with monotonically increasing or decreasing X values.

RYR *Y1 Y2 [X1 X2]*

'Restrict Y range'; throws away datum points in the current arrays which have Y values outside the limits Y1 and Y2. These limits apply to the entire arrays unless X1 and X2 are supplied, in which case the operation is carried out only within that X range.

SAVE *[file name or NDF name [n1-n2]]*

Dumps the contents of the STACK in a form suitable for subsequent reacquisition using `RESTORE`. The data can be in either NDF or "native DIPSO" format, depending on the flag established by the `USENDF` command. When specifying an NDF, do not include any file type. The default NDF name is 'SAVE_STK'. If not using NDFs, the default file name is 'SAVE.STK'.

It is possible to specify subsets of the stack for saving, but then a file or NDF name is mandatory. The entire contents of the stack are saved by default.

SCREENRD *[brkval]*

Input data from the terminal to the current arrays; terminate with "\". Each input line should contain a pair of X and Y values but nothing else. If a value of "brkval" is specified on the command line, then Y values matching it are assumed to flag gaps in the data.

SCROLLVT *[n1 n2]*

When a VT emulant is being used, this command result in text being scrolled between lines n1 and n2 only. Thus you can scroll text to, say, the bottom half of the screen (`SCROLLVT 14 25`) while plotting on the top half (`TZONE 5`). If specified, both n1 and n2 must be in the range 1 to 25, an at least two lines must be allowed for. Full screen scrolling results if no line range is given.

SHELL *command*

The supplied command is executed within a Bourne shell running in a child process which terminates when the command is completed. Note, the command is assumed to have failed if the status value returned by the command is non-zero. An alternative to using the `SHELL` command is to suspend the process running DIPSO by pressing control-Z, issue the required command, and then re-awaken the dipso process by typing "`fg`". The `SHELL` command is provided for use in DIPSO command files.

SL *[N1 [N2]]*

Stack list: gives the entry number, number of points, first and last X values, and the first characters of the title associated with each STACK entry. By default, the entire stack is listed (N1 = 1, N2 = no of stack entries), otherwise the first (N1) and last (N2) entries to be displayed can be specified. If N1=0, the contents of the 'current' arrays are also summarised. The "-" operator can be used to separate N1 and N2.

SLWR *[N1 [N2]]*

Stack List WRite. Operates identically to `SL`, but outputs results to a file STACK.LIS in the current directory, instead of to the terminal.

SM *Sigma*

Smooths Y data in the 'current' arrays with a Gaussian filter (FWHM=2.354*Sigma). `QSM` gives a much faster, and scarcely less accurate, result.

SNIP ***[X1 X2 X3 X4 X5 X6 ... X49 X50]***

Cuts out data from the 'current' arrays. Pairs of X values can be provided as optional parameters; if no parameters are given, the cursor is used to define regions to be SNIPped. Each pair of X values, whether input at the terminal or defined by cursor, must be in order of increasing X (though this constraint does not apply to successive pairs of values). To get out of `SNIP` mode when using the cursor, define X2<X1.

WARNING: `SNIP` *only* works successfully on data sets that have monotonically increasing X values. Remember, 'snipping' is done on data held in the 'current' arrays. Be careful not to plot STACK data and mistakenly think that it is the plotted data that are being edited. Furthermore, it is not possible to recall snipped-out data points, so it is wise to maintain an untouched data set on the STACK.

SP0RD ***file name or NDF name [logical]***

Reads a SPECTRUM format 0 file into the 'current' arrays. This is the format that data output from IUEDR are normally in (see SUN/37 for details); for other purposes, `READ` (and `WRITE`) are recommended. Maximum number of points allowed is 20,000. WORV is assumed to be 1.0.

The data can be in either NDF or "native DIPSO" format, depending on the flag established by the `USENDF` command. When specifying an NDF, do not include any file type.

If YES (the default) is supplied for the second (optional) logical parameter any zeros in the data file are treated as gaps in the data. If NO is supplied, any zeros are treated as good data values. This replaces the undocumented PANICRD command.

SP0WR ***file name or NDF name***

Outputs a SPECTRUM format 0 file (see the IUEDR documentation SUN/37 for a description of the SPECTRUM formats). But you should be using `WRITE`! The data output are those stored in the 'current' arrays.

The data can be in either NDF or "native DIPSO" format, depending on the flag established by the `USENDF` command. When specifying an NDF, do not include any file type.

SP1RD ***filename[.typ]***

Reads a SPECTRUM format 1 file (see SUN/37 for details) into the 'current' arrays. Maximum number of points allowed is 20,000. WORV is assumed to be 1.0. The default file type ([.typ]) is '.DAT'. This command cannot read NDFs.

SP2RD ***filename[.typ]***

Reads a SPECTRUM format 2 file (see SUN/37 for details) into the 'current' arrays. Maximum number of points allowed is 20,000. WORV is assumed to be 1.0. The default file type ([.typ]) is '.DAT'. This command cannot read NDFs.

SQRTX ***(no parameters)***

Replaces the X values in the current arrays by their square root (throwing away any negative values).

SQRTY ***(no parameters)***

Replaces the Y values in the current arrays by their square root (throwing away any negative values).

STATUS ***(no parameters)***

Returns information on the current status (device number, X and Y ranges, *etc.*).

TADD ***[string]***

Adds a string to the end of the current title.

TENX ***(no parameters)***

Replaces the X values in the current arrays by 10.0**X.

TENY ***(no parameters)***

Replaces Y values in the current arrays by 10**Y

TICKS ***[dx [dy [nx [ny]]]]***

Controls the appearance of tickmarks on plots. The arguments dx and dy control the spacing between major (labelled) tickmarks on the x and y axes, respectively; nx and ny control the number of spaces between major ticks (*i.e.* no. of minor ticks = nx/y minus 1).

A zero for any argument results in the relevant aspect of the plot design returning to automatic control; just typing `TICKS` (no arguments) returns to fully automatic tickmark design. Negative arguments result in no tickmarking.

If you supply grossly inappropriate arguments (*e.g.* trying to squeeze too many numbered ticks onto an axis) then you will certainly find the resulting plot not to your satisfaction. Beware, in particular, of forgetting to scale by appropriate powers of ten (which may be subsumed into the axis label). Unless you are very familiar with the range of the data being plotted, it is wise to do initial plots with autodesign in force.

TITLE ***[string]***

Change the title associated with data in the 'current' arrays. Null strings are accepted as such unless `TPROMPT` has previously been set TRUE (q.v. `TPROMPT`). The string must NOT contain any control characters (which would probably cause a crash in the graphics library routines). Since a comma is normally interpreted as the end of a string, it is necessary to enclose strings containing commas in double quotes; *e.g.*

```
>SPORD TEST,TITLE "Commas can, I think, be useful",PUSH
>SPORD TEST,TITLE I have no commas, PUSH
```

gives:

```
Commas can, I think, be useful
```

and

```
I have no commas
```

respectively. It's a good idea to develop the habit of using double quotes regularly, even if you don't use commas often in strings.

The price paid for being able to include commas in strings is that other usage of quotes in TITLE is forbidden. (Actually, it's not, but the rules are so complex as to effectively forbid use of quotes.)

TPROMPT ***logical***

If "logical" is T or Y, then the command TITLE will prompt for an input string if none is provided; if F or N (the default), it won't.

TLINE ***line***

Change line attributes (continuous, dot-dash *etc*). The index 'line' must be in the range 1-5 (1 = continuous lines). (See also TROT).

TOFLAMBDA ***(no parameters)***

If the X and Y data in the current arrays are in units of Hz and erg/cm2/s/Hz, TOFLAMBDA will convert to Angstroms and erg/cm2/s/A.

TOFNU ***(no parameters)***

If the X and Y data in the current arrays are in units of Angstroms and erg/cm2/s/A, TOFNU will convert to Hz and erg/cm2/s/Hz.

TOV ***wav0***

Convert X values from wavelength to velocity (the inverse of TOW), where Wav0 is the appropriate rest wavelength in Angstroms.

No changes are made to the Y array. This would normally mean that subsequent measurements made with EW or FLUX would be in rather strange units (*e.g.* km/s, or erg/cm2/s/[km/s]). To avoid this anomaly, an internal variable WORV (for Wavelength OR Velocity) is associated with each data set. This has the value 1.0 by default, but is reset to Wav0/c (where c is the speed of light in km/s) on using TOV.

(If you are reading in data sets with velocity as the X co-ordinate it is usually safest to convert to wavelength [TOW] then back to velocity [TOV] in order to obtain an appropriate WORV value. This is unnecessary if you use READ/WRITE or SAVE/RESTORE to move data in and out of DIPSO.)

WARNING: If you have set the X range, you'll probably need to change it to get anything on the plotting surface after using TOV or TOW!

TOW ***wav0***

Convert X values from velocity to wavelength (see TOV), and resets WORV to 1.0.

TROT ***(no parameters)***

Implements automatic rotation of line attributes. Each plot begins with the line style defined by the last use of TLINE (or style 1, if TLINE hasn't been called).

Switched off with NTROT.

TPORT ***Zn Xmin Xmax Ymin Ymax [WXmin WXmax WYmin WYmax]***

Defines a plotting subzone (no. "Zn", where 100>Zn>8). The dimensions of the subzone (Xmin...Ymax) are normalised such that the total dimensions available (regardless of device) are 0 to 1 in both axes. (The corresponding physical dimensions can be discovered using QAREA.)

In general, a plot consists of axes, axis labels, and a header, as well as the data. Technically, the plotting subzone is a "graph window"; within this graph window a "grid window" is present. The grid window is exactly filled by the axes, and so will normally be smaller than the graph window (to leave room for the labelling). DIPSO will normally work out default grid window dimensions, but you can define your own (WXmin...WYmax). This might be useful if you want contiguous axes in different plots, for example.

Just as the graph window is defined in terms of fractions of the available plotting area, so the grid window is defined in terms of fractions of the available graph window. Thus parameters WXmin *etc.* must also be in the range 0 to 1 (regardless of the graph window dimensions).

To access a given subzone, use TZONE.

TSTRIP ***(no parameters)***

Removes leading blanks from the title associated with the 'current' arrays.

TSWAP ***n***

Copies the title from STACK entry 'n' to the 'current' arrays.

TWEIGHT ***weight***

Alters the weight (*i.e.* 'heaviness') of data curves plotted with PM, on devices which support this feature. No other lines (axes, *etc*) are effected. The weight must be 1-5 (initial setting is 1). See also LWEIGHT.

TZONE ***zn***

Selects zone "zn" for plotting. The zone numbers are:

0: entire surface

1: top left quarter

2: top right quarter
3: bottom left quarter
4: bottom right quarter
5: top half
6: bottom half
7: left half
8: right half

Additional zones can be user-defined if required (see TPORT). When using such additional zones special care is needed to avoid overplotting previous zones (use ERASE to get a "page throw" on hard copy devices); this is taken care of automatically with zones 0-8.

UBVRD ***u b v [dx]***

Converts UBV magnitudes to fluxes, and stores the results in the 'current' arrays. If the U, B and/or V magnitude is unknown, 0 should be entered. (Should you want to actually input a value of 0, I'm afraid that you'll have to use something like 0.0001, or put in a value of (*e.g.*) 5 and then carry out some arithmetic using TENY, YMULT and LOGY).

The data are plotted at assumed wavelengths of 3600, 4400 and 5500 Angstroms using lines 2*dx wide (default value of dx: 50 Angstroms). Conversion from magnitudes to fluxes is carried out using

```
Mag = -C - 2.5*Log10(Flux)
```

where C is 20.94, 20.51 and 21.12 for U, B and V respectively. (These values are from the absolute flux measurements of Vega made by Tug et al, Oke & Schild, and Hayes & Latham, for V; and a normalised Kurucz Vega model for U and B).

USEHTX ***[logical] [showme]***

If 'logical' is Y, YES, T or TRUE, the HELP command will display help information in hypertext format, using a World-Wide-Web browser. If N, NO, F or FALSE is supplied for 'logical', help information will be displayed in plain text format in the DIPSO command window. If no value is supplied for 'logical', YES is assumed.

If supplied, 'showme' should contain a string giving the unix command used to run the Starlink showme utility. The supplied command string will be used to initiate hypertext help sessions until a new value is supplied. The command used initially is "<dir>/showme", where <dir> is the path to the directory containing Starlink executable files which was used when DIPSO was installed (typically /star/bin).

USENDF ***[logical]***

If 'logical' is Y, YES, T or TRUE, data accessing commands such as READ, WRITE, SAVE, RESTORE, etc, will use NDF structures (see SUN/33) to store data in. If N, NO, F or FALSE is supplied, they will use "DIPSO native" files as used by DIPSO prior to version 3.0 (but note, that these files will NOT be portable from one operating system to another). If no value is supplied for 'logical', YES is assumed.

USSPRD *filename[.typ] [epsmin]*

Reads data from the IUE 'Uniform' Low Dispersion Atlas, ULDA, which have been output using the USSP (see SUN/20). The filename extension defaults to '.ULD'. This command cannot read NDFs. Each point in the USSP spectrum has an associated error index, called epsilon, with the following meanings:

100 : No special conditions

-200 : Extrapolated at upper end of ITF

-220 : Microphonic noise

-250 : Filtered bright spot

-300 : Unfiltered bright spot

-800 : Reseau in extracted spectral region

-1600 : Saturated

-3200 : Not photometrically corrected

The epsilons are not necessarily reliable indicators of data quality. DIPSO rejects points on input if they are flagged with an epsilon less than or equal to 'epsmin' (which defaults to -251), leaving the spectrum in the current arrays. It is forbidden to set epsmin less than or equal -1600, since this would result in totally unflagged, certainly bad, data being acquired.

If epsmin is given a value greater than zero, then *all* datapoints are read into the next available stack entry, and the epsilon array into the subsequent stack entry. More subtle doctoring of the data is then possible, using `USSPCLIP` (q.v.). However, it is recommended that the default epsmin be accepted unless you really know what you're doing, and have good reasons to choose a different value.

USSPCLIP *epsmin n1 [n2 w1 w2]*

Clips points out of IUE USSP spectra which have 'epsilons' less than or equal to epsmin. The data are expected to have been previously read into the stack using the `USSPRD` command (with its epsmin parameter given a positive value); 'n1' is the stack entry of the flux data, and 'n2' (which defaults to n1+1) that of the epsilon array. The clipping is done over the wavelength range w1<w<w2 (default: full wavelength range).

VCORR *vel [mode]*

If mode=1 (the default), `VCORR` 'unshifts' X values back to a zero-velocity reference frame by replacing the values with

```
X2 = X1 / (1.0 + vel/C)
```

where C is the velocity of light in km/s.

If mode=2, `VCORR` applies a velocity shift to the data by changing the X values:

```
X2 = X1 * (1.0 + vel/C).
```

If the mode=1 or 2 the Y values are not changed. If mode=-1 or -2 the Y values are also adjusted such that $f_X dX$ is constant – *e.g.* if mode=-1 then

```
Y2 = Y1 * (1.0 + vel/C)
```

and similarly, *mutatis mutandis,* for mode=-2.

WRITE ***output file name or NDF name [model NDF name]***

Writes the contents of the 'current' arrays into a disk file suitable for subsequent re-reading using `READ`. The data can be in either NDF or "native DIPSO" format, depending on the flag established by the `USENDF` command. When specifying an NDF, do not include any file type.

If the name of an existing NDF is given for the second (optional) parameter then the specified NDF is used as a "model" for the output NDF. The output NDF is initialised to hold a copy of the model NDF (including all extensions). The data values in the current array are then inserted (if possible) into the output DATA array at their correct wavelength positions. This provides a mecahanism for creating NDFs which look like they have been created by other packages. For instance, if you want to use DIPSO to process the NDF "`my_dat`" originally created by the JCMTDR package, then you would use "`read` my_dat" to read it in as normal, and then you could use "`write` new_dat my_dat" to write the processed data to NDF "`new_dat`", copying all the JCMTDR extension information (etc) from the original "`my_dat`" NDF. The resulting NDF could then be put straight back into JCMTDR.

XABS ***(no parameters)***

Replaces the X values in the current arrays with ABS(X).

XADD ***c***

Adds a constant, "c", to the X values in the current arrays.

XDEC

Replaces the X values in the current arrays by [X−INT(X)].

XDIV ***c***

Divides the X values in the current arrays by a constant, "c".

XINT ***(no parameters)***

Replaces values in the current X array with INT(X).

XINV ***(no parameters)***

Replaces the X values in the current arrays by their inverse.

XMULT ***c***

Multiplies the X values in the current arrays by a constant, “c".

XNINT ***(no parameters)***

Replaces X values in the current arrays by NINT(X).

XSUB ***c***

Subtracts a constant, “c" from the X values in the current arrays.

XCORR ***n1 n2 [lolag hilag p]***

Cross-correlates the data set in stack entry n1 with the data set in stack entry n2. (Autocorrelation functions can be calculated by defining n1=n2.) Entry n1 contains the “stationary” data.

The range over which the cross-correlation function is evaluated is controlled by the parameters lolag and hilag, which must be in the same units as the stack entries. The default lag is given by:

```
lag = MIN{ (x[n]-x[1]), 100*(x[n]-x[1])/n }
```

where there are ‘n” datum points in stack entry n1, with X values from x[1] to x[n]. Then lolag defaults to -lag, and hilag to +lag. The parameter p is the fraction of each data set endmasked (at each end, with a cosine bell), and defaults to 0.05.

The cross-correlation is carried out in the units of the X arrays (which have to be monotonically increasing, in the same units for both data sets, and at least partially overlapping if you want sensible results). This is for generality. However, a typical application would be to find velocity shifts between data sets in wavelength space — but a velocity shift is a function of wavelength in wavelength space. The way to get round this is to take logs of both data sets (`LOGX`), then evaluate the correlation function. Then:

```
Delta(V) = [10**{Delta(LogLambda)} - 1] * C
```

where C is the speed of light (making sure that your units all match up). The arithmetic can all be done in DIPSO (`TENX`, `XSUB`, `XMULT`).

IMPORTANT: XCORR just does the cross-correlation; it is not a ‘black box’. I can’t think of a situation in which you shouldn’t first rectify your data (with `CREGS` and `PF`, or `CDRAW`, followed by `ADIV`) then subtract the continuum (`YSUB` 1), for both stack entries, in order to minimise edge effects.

XJ ***(no parameters)***

“Justifies” the X range — *i.e.* sets X limits to exactly match the range of the data being plotted (c.p. `XT`).

XLAB *[string]*

The label for the X axis; the default is 'Wavelength'. If the string contains commas it *must* be enclosed in double quotes (*c.f* `TITLE`).

XMAX *x*

Set the maximum X value for plotting to 'x'. Negated with `NX`.

XMIN *x*

Set the minimum X value for plotting to 'x'. Negated with `NX`.

XR *x1 x2*

Set the X range; x1 is the left-hand value for the plot, x2 the right-hand value. Negated using `NX`.

XREV *(no parameters)*

Reverses the ordering of the data in the current X arrays, maintaining X-Y pairing

XSORT *(no parameters)*

Sorts the X values in the current arrays into increasing values, maintaining X-Y pairing. Breaks in the data are lost.

XT *(no parameters)*

"Trims" the X range of a plot — *i.e.* sets X limits which are some integer multiple of the distance between tickmarks (c.p. `XJ`).

XV *(no parameters)*

Obtain X values using the cursor. Do a cursor hit at the same place twice to get out of `XV` mode.

XYSWAP *(no parameters)*

Swaps the contents of the X and Y arrays, maintaining the break arrays unchanged. (Use `CLRBRK` if this leads to unwanted results.)

XYV *(no parameters)*

Obtain X and Y values using the cursor. Hitting the same place twice exits `XYV` mode.

YABS *(no parameters)*

Replaces the Y values in the current arrays with ABS(Y).

YADD ***c***

Adds a constant, 'c', to the Y values stored in the 'current' arrays.

YDEC

Replaces the Y values in the current arrays by [Y−INT(Y)].

YDIV ***c***

Divides the Y values in the 'current' arrays by a constant, 'c'.

YINT ***(no parameters)***

Replaces Y values in the current arrays with INT(Y).

YINV ***(no parameters)***

Replaces the Y values in the current arrays by their inverse.

YMULT ***c***

Multiplies the Y values in the 'current' arrays by a constant, 'c'.

YNINT ***(no parameters)***

Replaces Y values in the 'current' arrays with NINT(Y).

YSUB ***c***

Subtract a constant, 'c', from the Y values stored in the 'current' arrays.

YJ ***(no parameters)***

"Justifies" the Y range — *i.e.* sets the Y limits to exactly match the range of the data being plotted (c.p. `YT`).

YLAB ***[string]***

Replaces the label for the Y axis; the default is 'Flux'. If the string contains commas it *must* be enclosed in double quotes (*c.f* `TITLE`).

YMAX ***y***

Sets the maximum Y value for plotting; negated with `NY`.

YMIN ***y***

Sets the minimum Y value for plotting; negated with `NY`.

YR ***y1 y2***

Sets the Y range for plotting; y1 is the lower value for the plotting frame, y2 the upper. Return to autoscaling with `NY` or `NXY`.

YT ***(no parameters)***

"Trims" the Y range of a plot — *i.e.* sets Y limits which are some integer multiple of the distance between tickmarks (c.p. `YJ`).

YV ***[Xvalue]***

Obtain Y value. If an X value is supplied, then the corresponding Y value is obtained (by linear interpolation) from the data in the current arrays. Otherwise, the graphics cursor is brought up to permit Y values to be measured from the terminal; hit the same place twice to get out of this mode.

YXN ***power***

Replaces values in the 'current' Y arrays by Y*(X**power). (Some people like to plot data as F*[Lambda**4], for example.)

ZANSTRA ***line F(obs) T(neb) [E(B-V)]***

Calculates a (black-body) Zanstra temperature, using the observed flux, F(obs)
(in erg/cm2/s), of a recombination line. The lines for which this calculation can be performed are (H I) 4861; (He I) 4471, 5876; and (He II) 1640, 4686. The 'line' parameter is the wavelength, in Angstroms, of the selected line.

The calculation requires the location of a 'continuum' point longwards of the ionization edge of the ion concerned. This is obtained from the cursor; thus a plot, in erg/cm2/s/A vs Angstroms (or log10 thereof) is mandatory. Moreover, for a valid result the cursor hit has to correspond to the dereddened stellar flux; judicious prior use of `NEBCONT`, `ASUB` and `DRED` may therefore be useful.

T(neb) is the electron temperature of the ionized nebula, and may be input in units of K or 10,000K. This parameter is required because of the (fairly weak) temperature dependence of the ratio, R(line), of the effective recombination coefficients to the ion and line concerned (for a discussion of the physics in the Zanstra method, see *e.g.* Osterbrock, 'Astrophysics of Gaseous Nebulae'). The adopted temperature dependences of R(line) are:

```
R(1640) = 2.00 x (t**0.10)
R(4471) = 19.61 x (t**0.27)
R(4686) = 4.37 x (t**0.29)
R(4861) = 8.49 x (t**0.06)
R(5876) = 5.39 x (t**0.39)
```

where the parameter t = T(neb)/(10,000K).

E(B-V) is used to deredden the line flux (*only — i.e. not* the continuum flux); a galactic reddening law with R=3.1 is adopted. If this is inappropriate to your data, deredden F(obs) to your own prescription and use E(B-V)=0 (which is the default value).

The output from this command consists of the Zanstra temperature and a normalising constant, C(norm.). The latter quantity is the number by which a black-body spectrum calculated using `BBODY` must be multiplied (YMULT) to make it pass through the (X,Y) co-ordinates selected using the cursor.

A.2 Finally...

Congratulations on reading this far (unless you've cheated, and skipped straight to the end...). Features, bugs, complaints and comments should be addressed to the Starlink support mailing list (`starlink@jiscmail.ac.uk`); but please check the documentation first!

B History

This section records the changes introduced with each new version of DIPSO. *NOTE*, earlier changes may be over-ridden by later changes.

B.1 Changes introduced by DIPSO V3.6-5

- The ELFINP command now has a parameter that can be set non-zero to allow input to be read from the currently running script file rather than from the keyboard.

B.2 Changes introduced by DIPSO V3.6-4

- The dipso_link command has been modified in order to avoid problems on Solaris.

B.3 Changes introduced by DIPSO V3.6

- The `READ` command has been modified so that it will attempt to read spectral axis information from the WCS component of an NDF if the NDF was not created by DIPSO. If the NDF has no WCS component, the spectral axis information will be read from the NDF AXIS structures, or the FITS header cards in the NDF FITS extension (if any). Details of the spectral axis are displayed.
- The `WRITE` command has been modified so that it will add a WCS component to the output NDF describing the spectral axis. This allows it to be used by other Starlink applications such as SPLAT, KAPPA, etc.

B.4 Changes introduced by DIPSO V3.5-6

- A bug has been fixed which caused the ELFRESTC command to be unable to open files which contain any lower case letters in their file name.

B.5 Changes introduced by DIPSO V3.5-5

- A bug has been fixed which caused the ISATM command to be unable to find entries in the ATOMIC.DAT file when being run under Linux.

B.6 Changes introduced by DIPSO V3.5

- The new command tt RDCAT has been added to enable data held in FITS table format to be read.
- The maximum number of clouds which can be specified using ISINP has been increased from 9 to 18.
- A bug has been fixed which prevented ELFOPT working correctly.

B.7 Changes introduced by DIPSO V3.4

- The USEHTX command has been modified to allow the path to the Starlink showme utility to be given explicitly. This need only be done if the HELP command fails to find the showme utility (for instance, if is not on the user's current PATH, or if it is executed by means of an alias).
- The contents of restored stacks held in NDF structures are now listed as they are restored.
- The makefile has been modified so that "my_dipso" works as described in section 8.2 (i.e. you no longer need to set the INSTALL environment variable prior to using "my_dipso").
- The new command ENV has been added, which displays the value being used by DIPSO for a named environment variable.
- The URL for the DIPSO WWW home page has been corrected.
- Default values for environment variables such as DIPSODIR can now be supplied on the DIPSO command line. They should take the form of a comma separated list of "name=value" pairs. These values are only used if the corresponding environment variables are not defined.
- The command prompt used within DIPSO can now be specified using the environment variable DIPSOPROMPT.
- Various changes to the Fortran code have been made to allow DIPSO to be compiled using the g77 compiler under Linux (which requires closer adherance to the ANSI Fortran-77 standard than other copilers). None of these changes should produce any noticable change in behaviour.

B.8 Changes introduced by DIPSO V3.3

- The internal array sizes have been increased. The current arrays can now hold 200,000 points, and the stack arrays can now hold 800,000 points in up to 200 stack entries.
- Hypertext documentation and help have been included through the HELP and USEHTX commands, and a hypertext version of SUN/50.
- The SHELL command has been re-instated, which allows system commands to be run from within DIPSO without the need to press control-Z.
- The functionality of the COMMAND command has been enhanced to provide faciltities for listing commands which do particular jobs, and which contain specified keywords in their descriptions.
- NDF accessing is now done using latest NDF facilities to allow transparent access to foreign data formats (IRAF, FITS, etc, see SUN/55 and SSN/20).
- All uses of NAG routines have been replaced by equivalent public domain algorithms (in fact the only command which used NAG was PF).
- The help.lis file containing plain text descriptions of all DIPSO commands has been updated to be consistent with SUN/50. This file is used by the HELP command.

- Re-direction of GKS error messages removed. They should now go to the screen.
- Bug corrected which caused error messages to be displayed when PUSHing the current array if the last point in the data array was bad.
- ELFIT changed to avoid floating exception if the variances come out negative (a warning is issued and zeros are used instead).
- ELFSAVEC now offers the user the chance to overwrite an existing file if one exists. Also, it no longer converts the supplied file name to upper case.
- SM bug corrected which caused extension of smoothing domain beyond the upper wavelength limit, and could potentially cause completely spurious results if the upper wavelength limit was at the end of the array.
- Organisation of source files and object files changed to use a sigle library.

B.9 Changes introduced by DIPSO V3.2

- A new command TWEIGHT has been introduced to allow control of the weight used to draw data curves independantly of the weight of other lines.
- Bug corrected which caused the PUSH command to fail with a message like:

```
PUSH: breaks/data mismatch
number of breaks:125
index of last break:22792
number of points:22791
Command line aborted
```

  after reading in a spectrum using SPORD. This only occurred if the first and/or last data point in the spectrum contained the value zero.
- Bug corrected which caused the NEBCONT command to re-issue a prompt for a value, rather than accepting the default value as read from the specified file, if <RETURN> is pressed (with mode1,2,3=0).
- Error messages generated by GKS are now displayed as they occur, rather than being stored up and displayed altogether when DIPSO is exited.
- Bug fixed which caused cursor operations (eg XV, SNIP, etc) to fail after an invocation of the condition handler caused by floating point exceptions, control-C's etc.
- Shell meta-characters can now be included in responses given to prompts for NDFs.

B.10 Changes introduced by DIPSO V3.1

- Command line recall and editing is now available.
- Condition/signal handling is now available on both VMS and UNIX. The HANDLER and CRASH commands have been re-instated.

- The NDF accessing layer has been completely re-written to eliminate the many bugs related to the reading and writing of NDFs in the previous version DIPSO. The description of how to write a program which can create NDFs suitable for use with DIPSO (see section8.5.1) has been re-written.
- Various other bugs have been cleared up.
- The user can now decide whether to use NDF data files or "native DIPSO" data files. This is accomplished using the `USENDF` command. Commands `READ`, `WRITE`, `SAVE`, *etc.*, will now use the selected data format, and so the commands OREAD, OWRITE, OSAVE, *etc.*, are no longer needed and have been withdrawn.
- New commands added: `USENDF`, `REPORTING`, `TSWAP`, `TSTRIP`
- Commands re-instated from previous versions: `RECORD`, `REPLAY`, `RECALL`, SHELL (only on VMS),
- The `EXPAND` command now takes an optional second string parameter which specifies which components of the plot are to be expanded.
- The behaviour of prompts has been unified (to some extent). Giving "!" in response to any prompt will cause the current command to abort and return you to the main DIPSO command prompt. Giving "!!" will abort the command and also abort DIPSO (after saving the stack to EXIT_STK.sdf or EXIT.STK). Some prompts now include a suggested default within the prompt string which will be used if a null value is supplied by the user.
- `READ` command (when using NDFs) will now read NDFs not written by DIPSO (*i.e.* ones which don't have a DIPSO_EXTRA extension).
- `WRITE` command (when using NDFs) now has an optional second string parameter which specifies the name of a "model" NDF on which to base the output NDF created by the `WRITE` command.
- SP0RD command (when using NDFs) now has a second (optional) YES/NO parameter which determines if zeros in the data file are treated as gaps in the data.
- `SCREENRD` command, "\" is now used to end input on VMS and UNIX.
- The VMS version is no longer compiled "/DEBUG/NOOPT" so hopefully it could be a bit (!) faster?
- VMS and UNIX versions of source files have been unified.

B.11 Changes introduced by DIPSO V3.00

The default file format for 'unformatted' DIPSO files (SP0) has changed from native unformatted format to STARLINK NDF format. This allows the transport of files between different machines without translation being necessary. It also means that files generated by DIPSO can be automatically input to all standard STARLINK packages.

This version of DIPSO is the first multi-platform release. It has been tested on VAX/VMS, DECstation, and Sun Sparcstation machines. Due to the different way system variables are provided by the two operating systems DIPSO has been enhanced to support both the VMS

logical name syntax, and the UNIX environment variable syntax. Thus the following two filename specifications are equivalent and accepted by DIPSO on all platforms.

- VAX logical name form — `OWNERDIR:my_data_file`
- UNIX environment form — `$OWNERDIR/my_data_file`

UNIX users should note that filenames are ALWAYS CaSe sensitive.

B.12 Changes introduced by DIPSO V2.00

(If you didn't use DIPSO before 1987, pass over this section. If you are an old hand, then: the more experienced you are, the more important it is that you should *read this section carefully!*)

This release of DIPSO is a fairly extensive revision of earlier versions. In particular, the graphics have been converted to the GKS standard (mainly by JM), interstellar line analysis has been included (`IS...` commands), and various aspects of Fourier analysis are now possible. Because interstellar line profiles are now most easily computed from within DIPSO, the old `BACHRD` and `BACHWR` commands are no longer documented, and `DIPSODIR:ATOMIC.DAT` has been extensively revised (`$DIPSODIR/ATOMIC.DAT` on UNIX machines). `ALASRD/ALASWR` have been preserved, however, as the simplest way of getting data in and out of DIPSO. (`ALASRD` has actually been updated to allow more general inputs.)

In an attempt to (partially) rationalise the command names, and make it easier to locate groups of related commands in the reference section, some command names have been changed. In particular, the old `ELF` commands are now all prefixed by ''`ELF`'' (surprise!). (The minus side is that you'll have to learn the new command names; but the plus side is that `ELF` now carries out an error analysis for you.) The two-spectrum arithmetic functions are now renamed `ADIV, AADD` *etc.* (from `DIV, ADD` *etc.* to avoid the trap of typing, say, `ADD 3` in the expectation of adding 3 to the current arrays (the "A" prefix stands for "array".) Some one-spectrum arithmetic functions have also had their names changed to a more uniform scheme; *e.g.* `XSH` has become `XADD`, `CMULT` has become `YMULT`, *etc.* The old command names will still work in some cases, but are not recommended. You are urged to read right through the new documentation for individual commands, as many other minor modifications have been made, and new functions added.

The most important changes: you will probably find out quite quickly that the default file extension for command files has changed to `.CMD`.

The following commands have *changed default functionality*, and you should therefore check them especially carefully: `ALASRD`, `ALASWR`, `DRED PWRITE`, `TPORT`, `READ`, `WRITE`, `SAVE`, `RESTORE`.