GPLOT: A **DIMFILM** Based Graph Plotting Program for CDC NOS 2.8

Nick Glazzard

April 9, 2015

1 Introduction

GPLOT is an interactive graphics program primarily focussed on graph plotting from data files. It is based on U.L.C.C. **DIMFILM**, a substantial graphics library written and maintained by Dr. John Gilbert between 1973 and approximately 1993. The version of **DIMFILM** used is from somewhere around 1982 and is the second major version, which moved away from CDC Fortran to portable, standard Fortran-77 (necessitated by U.L.C.C.'s move to Amdahl IBM compatible machines).

GPLOT, in contrast, is deliberately written for the CDC `FTN5` compiler under NOS 2.8 and is not intended to be portable. It uses seven letter identifiers, dynamic memory allocation (via the Common Memory Manager), and makes free use of NOS system calls. In spite of this, there is a Unix port, tested on Mac OS X, which might make preservation of **DIMFILM** more likely in the long run.

In overall intent, **GPLOT** is like a very stripped down Gnuplot. A particular limitation is that it can only plot the contents of data files. It cannot evaluate functions itself. **GPLOT** relies on character string parsing subroutines written by Dr. Adrian Clark circa 1985. Dr. Clark also wrote a **DIMFILM** device driver for Tektronix storage tube terminals on which the **GPLOT** Tektronix and, to some extent, **GTerm**, support is based.

The primary output device for **GPLOT** is **GTerm** – a simple terminal emulator with colour graphics capabilities written in Python (see the **GTerm** documentation for more information). **GPLOT** also supports, to some extent, Tektronix 401x terminals (which `xterm` emulates), and fully supports output to something very close to Encapsulated PostScript files.

GPLOT is currently at version 0.1 and should be considered as under development. The basic facilities needed to plot graphs (and other graphics) are there, but there are many things that could usefully be added and there may be things that need adjusting for maximum convenience as experience is gained of using it. That said, it is unlikely that this author will be adding anything further to it.

A major goal for **GPLOT** is that it should be *very easy to use* — as easy as other graph plotting programs on modern operating systems (such an Gnuplot, matplotlib, etc.) — although **GPLOT** will never rival the full capabilities of those systems.

To install **GPLOT** on an emulated CYBER mainframe (with dtCYBER), please refer to Section 6.1.

GPLOT was written in late 2013. A 'package' that might be installable fairly easily was put together in March 2015. This 'package' includes **DIMFILM**, support libraries and fonts as well as **GPLOT**, in both source and binary form, together with batch jobs to rebuild the binaries from the source.

2 Running GPLOT

GPLOT needs two files: `ABGPLOT` (the pre-loaded binary executable) and `DADIMFO` (which contains the font data for **DIMFILM**). To run **GPLOT** use:

```
/ATTACH,ABGPLOT.
/ABGPLOT.
```

`ABGPLOT` will `ATTACH` the font file (`DADIMFO`) itself.

The following control card options are recognized.

- `OBEY=FILE`
  GPLOT will start reading commands from the specified `LOCAL` file with name `FILE`. The current version exits when the file has been read, but this might change in future versions.

- `PARM=STRING`
  Used with `OBEY=FILE`. `STRING` is passed as one or more parameters to the `OBEY` file. See 3.5 for more details. Under NOS, if `STRING` needs to contain spaces (i.e. there is more than one parameter), it should be enclosed in $ signs. If any single parameter in `STRING` itself needs to contain spaces, that parameter should be enclosed in single quotes.

- `GET=YN`
  Turn on automatic `GET`s of indirect `PERMANENT` files before `READ` and `OBEY` try to access the files they need. `YN` is `YES` or `NO` (or an abbreviation down to `Y` or `N`). This saves having to issue `GET` in **GPLOT** (or NOS before **GPLOT** is run) in order to make the files to be used `LOCAL`. On the other hand, if changes have been made to a `LOCAL` version of a `PERMANENT` file, these will be lost when **GPLOT** `READ`s or `OBEY`s that file with 'auto `GET`' in effect. So be careful!

- `DEBUG=YN`
  Outputs debugging information (mostly to do with `OBEY` file arguments and parameter substitutions currently).

- `QUIET=YN`
  Displays the unabbreviated command and `OBEY` nesting level for every command executed. Note that this may upset some output devices (it seems).

An example of a **GPLOT** control card is:

```
ABGPLOT,OBEY=DOPLOT,PARM=$X Y 'A TITLE'$.
```

3 Commands

All commands can be abbreviated so long as they uniquely identify a command. (If the abbreviation is ambiguous, you will get a warning message and no command will be executed).

All commands are shown in `UPPER CASE`, as that is the primary (`NORMAL`) character mode for NOS. In `NORMAL` mode, lower case is equivalent to upper case, so lower case can be used if desired. **GPLOT** should *not* be used in `ASCII` mode (in which lower case characters are *really* lower case). That does not mean that lower case text cannot be plotted – it can.

Commands are divided in to related groups below, but that is only for convenience.

3.1 Basic Graphics Commands

These are the lowest level facilities (below graph plotting).

- `DEVICE NAME [OUTPUT-FILE]` — Select the output device.
  The EPS devices need an output file name to write to. Note that this will be created as a `LOCAL` file and will need to be made permanent with `SAVE` or `REPLACE`. See Section 3.4 for the available devices and their characteristics.

- `CLEAR` — Clear the drawing area.
  The device display will be set to its default background colour. Any previously drawn material will be erased. This is also required to flush output to files for the `EPSCOL` and `EPSBIN` devices.

- `BOUNDS XL XH YL YH` — Set the plotting bounds.
  This establishes a user coordinate system with x_l, y_l at the bottom left and x_h, y_h at the top right. The `MOVE`, `DRAW`, `PANE` and `BLANK` commands use this coordinate system.

- `PANE XL XH YL YH` — Set the `PANE` (clipping area).
  All drawing with `MOVE`, `DRAW` and `TEXT` will be clipped to the area specified here. In contrast, all graph plotting commands will apply inside this area, so that graphs will be scaled to fit this region. This is a way to plot multiple graphs side by side.

- `UNPANE` — Stop using any pane.
  The full drawing area will be used after this.

- `PANEOUTLINE` — Outline the pane area.
  A rectangle is drawn that matches the pane area.

- `BLANK XL XH YL YH` — Set the blank area.
  No drawing with either the basic graphics or graph plotting routines will make a mark inside the specified area. This can be useful for reserving a space for a graph key, for example.

- `UNBLANK` — Stop using any blank area.
  The previous blank area is no longer protected after this.

- `BLANKOUTLINE` — Outline the blank area.
  A rectangle is drawn that matches the blank area.

- `COLOUR R G B` — Set the RGB colour to use for drawing.
  All lines (and all the graphical elements are lines) will be drawn in this colour.

- `WIDTH WIDTH` — Set the line width.
  The width is a multiple of a device dependent "base width" and not all devices can draw wide lines (currently, only the Tektronix device cannot).

- `STYLE STYLENAME` — Set the line style.
  The following styles are defined: `SOLID`, `DASH`, `DOT`, `DASHDOT`. These are probably self explanatory!

- `MOVE X Y` — Move to position. The virtual pen is moved to (x, y) in `BOUNDS` coordinates.

- `DRAW X Y` — Draw to position.
  The virtual pen is lowered and moved to (x, y) in `BOUNDS` coordinates.

- `TEXT "TEXT"` — Draw text.
  The string `TEXT` is drawn at the current pen position. The string may be enclosed in double quotation marks to allow spaces in the string. The text will be drawn with the default font. In this version of **GPLOT**, there is no way to change fonts, although that would be fairly easy to add. The string may contain many special formatting sequences. The most frequently useful are `*L` to switch to lower case and `*U` to switch back to upper. The full set of format controls are described in the **DIMFILM** manual, although the use of `$` signs may be problematic.

- `FILL` — Fill the drawing area with the current colour.
  This clears the graphics area to the current colour, erasing it. It differs from `CLEAR` in that `CLEAR` always fills with white and empties any buffered graphics commands in the device (if relevant). `FILL` is only really useful after a `CLEAR` though, as it destroys anything already drawn. This command is device dependent and isn't useful for all devices. Only **GTerm** supports it currently.

3.2 Graph Axis Commands

Before plotting a graph, **GPLOT** needs to know the range and type of the axes on which to plot the graph. The default settings of `XYAUTO`, `XLINEAR` and `YLINEAR` mean that *none* of the commands below *need* to be used before plotting, but you may want to use them.

- `XYAUTO` — Find both axis ranges automatically.
  DIMFILM will examine the range of the data in use when an `XYPOINT`, `XYLINE` or `XYHISTOGRAM` command occurs and will set the axis ranges to match. Linear axes will be used.

- `XYSAME` — Keep the previous axis ranges.
  If a graph was plotted with `XYAUTO` on, and other graphs need to be plotted on the same axes, use `XYSAME` after the first graph has been drawn.

- `XRANGE XLO YHI` — Set the X axis range.
  The X axis will run from `XLO` to `XHI`.

- `YRANGE YLO YHI` — Set the Y axis range.
  The Y axis will run from `YLO` to `YHI`.

- `XLINEAR` — Use a linear X axis.
  This is the default.

- `YLINEAR` — Use a linear Y axis.
  This is the default.

- `XLOG` — Use a logarithmic X axis.

- `YLOG` — Use a logarithmic Y axis.

3.3 Graph Plotting Commands

These are the commands that get the data to be graphed and actually draw graphs. Before data can be plotted, it must be `READ`, so the graph drawing commands (`XYPOINT`, `XYLINE` and `XYHISTOGRAM`) will not work until a `READ` has got some data. The graphs that can be drawn depend on what data is `READ` — this can be two real numbers per point (for (x, y)), three – which adds information for a symmetric Y error bar, or four – which adds data needed for asymmetric Y error bars, or symmetric Y and X error bars. The graph drawing commands will always use all the data available, so if `READ` has got data for (x, y) and a symmetric Y error bar, for example, that is what `XYPOINT` and `XYLINE` will draw. To stop drawing unwanted items, simply re-read the data omitting the undesired items.

- `MAXPOINTS NUMBER` – Set maximum number of data points.
  Memory for data to be plotted is allocated dynamically (this could be done in most major Fortran implementations, even though it is completely non-standard in Fortran-77 and earlier — there is much, possibly deliberate, ignorance on the part of academics of what could actually be done in this language). Four real numbers are allocated for each point — two for an (x, y) coordinate and two for error bar data. On starting, **GPLOT** allocates space for 1000 points. This can be changed at any time (although any data `READ` will be thrown away if it is changed, of course). The number of data points is limited by the amount of central memory on the machine, and how much of that you are allowed to use by your account settings. Under emulation, there is no reason not to use the maximum available — 262,144 words of which 131,072 can be used by any one job (minus a bit, of course!). **GPLOT** is, unfortunately, not small ... and will grow as more facilities are added. At present, the maximum number of points allowed is set to 13000 and this is close to the maximum possible. The limit of addressable central memory is the main limitation on what can actually be done under NOS. Although extended memory and 'out-of-core' solutions (where most of the data resides on disk) may be possible in some cases, that isn't always so (e.g. **DIMFILM** needs the data it plots to be in central memory), and these methods always come with a significant performance and complexity cost.

- `READ NAME XCOL YCOL [YECOL [XECOL]]` — Read a data file using the specified 'columns'.
  The file `NAME` (which must be a `LOCAL` file – possibly obtained inside **GPLOT** using the `GET` command) should contain numbers (in the standard character code), with at least one per line, separated by spaces or commas. There may be any number of spaces preceding or separating each number (subject to line length limitations – the limit may be 80). Blank lines are skipped and ignored. Lines with the character `'C'` in column 1 are ignored (allowing comments). The `numbers` may be integers or any real number format (exponential or not),

optionally preceded by a + or − sign. `READ` will try to interpret anything that *could* be a number *as* a number. It will stop reading a data item when a non-numeric character is found — but this is not treated as an error.
Each space (or single comma) separated number on a line is termed a 'column', and these are numbered starting at 1. So column 2 is the second number on each line. Column numbers must be specified for `XCOL` and `YCOL` data to obtain at least an (x, y) coordinate for each point. As a special case, `XCOL` may be 0, in which case, the X coordinate is set to the point number (starting at 1). This allows a data file with a single data item per line to be processed.
If a column number is supplied for `YECOL`, then that column's values will be used for symmetric Y error bars. If a column number is also specified for `XECOL`, then that data will be used as either a symmetric X error bar or as the upper part of an asymmetric Y error bar, depending on the setting made with the `ASYMYERRORBARS` command. The default is to interpret `YECOL` and `XECOL` as the lower and upper extents of asymmetric Y error bars, as this may be more frequently useful, There is currently no way to draw asymmetric X error bars — although this could be added if the storage of 6 numbers per point was acceptable (I'm not sure it is).
If `NAME` is `HERE` (i.e. the string `HERE`), then no data file is opened. Instead, `READ` tries to read data from the command input stream itself (`INPUT` or an `OBEY` file – see below). `READ` will stop reading data items when it finds a line that contains the string `EOF` as its only item. This facility is mostly to allow other programs to write a self contained `OBEY` file that executes a complete plotting task then exits **GPLOT**. A major anticipated application of this is plotting data from APL.

- `MARKER NUMBER` — Set the marker number to be used for points.
  This is a character number in `Font 9001` (see the **DIMFILM** Revived manual for details). Characters are associated with numbers 2 to 25, but with 5 and 11 omitted. Numbers 2, 3, 4, 8, 9, 19 and 20 are perhaps the most useful. Something may need to be done to improve this font or at least supply descriptive names for the markers!

- `XYPOINT` — Draw an XY graph with points.
  The points will be drawn with the last marker set with a `MARKER` command, or 3 (the default) if no `MARKER` command is used. Depending on the data items `READ` and the `ASYMYERRORBARS` setting, symmetric Y, asymmetric Y or symmetric Y and X error bars will also be drawn.

- `XYLINE` — Draw an XY graph with lines.
  By default, data points will be joined with straight lines (linear interpolation – of a kind) and in `SOLID` line style. The line style can be changed with the `STYLE` command. Cubic or quintic interpolation can be used between the data points to get smooth curves instead of straight lines. At least 3 points are required for cubic and 5 for quintic interpolation. If there are insufficient points, linear interpolation is used as a fall-back. The interpolation is as supplied by **DIMFILM** — as always, whether it is appropriate depends on the data and its desired interpretation. I'm not certain exactly how **DIMFILM** comes up with the interpolation, although it is visually fine.

- `XYHISTOGRAM` — Draw an XY histogram.
  Bars (of a type determined by `HISTSTYLE` — default: `ABUT`) are drawn centered on the X

coordinate and extending vertically from the X = 0 axis to the Y coordinate. Error bar data (if any) is ignored.

- `HISTSTYLE STYLENAME [WIDTH]` — Sets the histogram bar style.
  `STYLENAME` is one of the following:
  - `ABUT` — Open rectangular bars are drawn, centred on an X coordinate, and with a width determined by the X location of the preceding point (apart from the first point, which uses the location of the second point), so that the bars touch one another along the X direction.
  - `ABUT+SHADE` — as `ABUT`, but the bars are filled with 'shading' — lines at 45 degrees spaced 0.02 of the X `BOUNDS` range apart.
  - `LINES` — Zero width lines are used for the bars.
  - `WIDE` — Bars of the specified `WIDTH` (in X axis units) are drawn without shading.
  - `WIDE+SHADE` — As `WIDE`, but with shading.

- `INTERPOLATE STYLE [N]` — Set the interpolation mode for `XYLINE`.
  `STYLE` may be: `LINEAR`, `CUBIC` or `QUINTIC` — which are hopefully self explanatory! As explained in `XYLINE`, either straight lines or smooth curves can join the points. If `CUBIC` or `QUINTIC` is used, the number of intermediate (linear) segments drawn for the curve between each point is controlled by `N` (default: 10).

- `ASYMYERRORBARS MODE` — Use asymmetric Y error bars if `MODE` is `ON`.
  If `OFF`, symmetric Y and X error bars are drawn. Error bars will only be drawn if error bar data has been `READ`, and both `YECOL` and `XECOL` must have been specified to get asymmetric Y error bars or symmetric Y and X error bars.

3.4 Graph Annotation Commands

Graphs almost always need some annotation. Minimal annotation is: ticks on the axes, numeric values against those ticks, and a framing rectangle drawn around the graph plotting area. These are always drawn after each graph drawing command (`XYPOINT`, `XYLINE`, `XYHISTOGRAM`) unless `ANNOTATE OFF` is used. `TITLE`, `XLABEL`, `YLABEL` and `GRID` annotation elements are only drawn if these are explicitly specified.

- `ANNOTATE ON` or `OFF` — Turn annotation on or off.
  Typically, annotation is wanted whenever a single graph is drawn. However, after drawing one curve (etc.) with annotation, it might be desirable to turn off annotation when drawing the other curves on the same axes. It is certainly more efficient to do so, and some devices with some line width settings may show visible changes when items are drawn over existing identical elements (for whatever reason).

- `TITLE "TEXT"` — Set the title.
  This is drawn at the top of the graph, centred on the width of the graph drawing area. `TEXT` may be enclosed in double quotes to allow spaces and format commands (such as `*L` and `*U`) are understood.

- `XLABEL "TEXT"` — Set the X axis label.
  This is plotted below the X axis (at the bottom of the graph drawing area.

- `YLABEL "TEXT"` — Set the Y axis label.
  This is plotted (vertically) to the left of the Y axis.

- `GRID STYLE` — Draw a grid with lines at the major axis ticks for one or both axes.
  Style may be: `NONE`, `X`, `Y`, or `BOTH`, which are hopefully self explanatory.

3.5 GPLOT System Commands

These commands don't draw anything, but they show the state of **GPLOT**, interact usefully with NOS, and allow something like plotting macros (with parameters) to be used.

- `OBEY NAME [PARAMETERS]` — Start reading commands from file `NAME`.
  The file may contain any **GPLOT** command (one per line), including `OBEY`. `OBEY` files may be nested up to 5 deep (this could be increased, but 5 seems likely to be enough). If `PARAMETERS` is specified, it should be a single string without spaces or a string enclosed in double quotation marks. Each space separated word in the `PARAMETERS` string defines a single parameter, which are given numbers based on position starting at 1 on the left. Up to 9 parameters can usefully be set up in the `PARAMETERS` string. When reading commands from an `OBEY` file, **GPLOT** will apply a simple parameter substitution process: each space separated word of the form: `$N`, where `N` is `1` to `9`, will be replaced by the `N`th word of the `PARAMETER` string. It is possible to pass parameters containing spaces by enclosing them in single quotes in the `PARAMETER` string. For example:

  ```
  OBEY FRED "FIRST 'SECONDA SECONDB'"
  ```

  defines two actual parameters that will be substituted for the formal parameters `$1` and `$2` wherever these occur as space separated words in the file FRED. `$2` will, in fact, be substituted as:

  ```
  "SECONDA SECONDB"
  ```

  in order to preserve the embedded space(s). This is useful for passing `TITLE` strings to `OBEY` files, for example.

- `HELP` — Outputs summary help information, including the supported output devices.

- `STATUS` — Displays various **GPLOT** state information.
  This includes the number of points available, whether anything has been read, the `BOUNDS`, whether any `PANE` or `BLANK` is in effect, etc.

- `LOGFILE NAME` — Opens a command log file called `NAME`.
  This is a `LOCAL` file to which any interactive input from you, the user, is written. Commands executed from any `OBEY` files are not logged. This lets you log a session and repeat it (perhaps after editing) by executing the log file as an `OBEY` file. You will need to `SAVE` or `REPLACE` the file to make it `PERMANENT` if you want to keep it between jobs. When `LOGFILE` is executed, any previously open log file is closed and a new one is opened. If the new one has a different name from the old, the old one will not be destroyed.

- `MEMTEST` — Test dynamic memory is working.
  Which it will be! More usefully, it also generates test data in the form of a sine wave, which can then be plotted to evaluate how well an output device is working, or to get familiar with **GPLOT**.

- `GET NAME` — Get an indirect access `PERMANENT` file.
  This is useful to avoid having to exit **GPLOT** to make some data file `LOCAL` so it can be `READ`. There is no command as yet to `ATTACH` a direct access `PERMANENT` file, although it would be easy to add one.

- `EXIT` — Exit **GPLOT**.
  This closes any log file, flushes graphics commands to the output device, and shuts down **DIMFILM**. Then it exits **GPLOT** — surprise!

4 Output Devices

GPLOT supports the four devices currently supported by **DIMFILM**.

- `GTERM` — **GTerm** colour graphics terminal.
  This is a Python program I wrote to allow APL to be used with its proper character set, as well as to function as a better alternative to an `xterm` emulation of a Tektronix 401x for vector graphics output (e.g. that from **GPLOT**). Unlike Tek 401x emulations, it supports colour, anti-aliasing and anything displayed can be saved as SVG. **GTerm** runs on Ubuntu Linux 12.04 and 14.04 and on Mac OS X 10.9 (Mavericks) and 10.10 (Yosemite). It is based on PyQt, PyOpenGL, PyCairo and quite a few other things[1]. On Ubuntu Linux, these packages must be installed before **GTerm** will run. On Mac OS X there is a binary version of **GTerm** with an installer and this should be as easy to install as any other Mac OS X app.

- `TEK4K` — Tektronix 4014 graphics terminal emulation.
  This should work with any Tek 401x emulator, although it has only been tested with the `xterm` version that comes with Ubuntu 12.04. Unfortunately, this version is rather too close an emulation of a real Tek 401x! All the output goes to a graphics 'display' — there is no separation of text and graphics which seems to be available (from what is posted on the Internet) in some other emulations. Although I can't actually see how that would be possible. . . It would be nice if it were, though, because having **GPLOT** commands and other textual output appear superimposed on graphics (if you are not very careful) is not great. **GPLOT** uses 'auto clear' to try to avoid this, in which every command causes the screen to be cleared before any other output appears. This makes interactive plotting of multiple graphs on a single set of axes impossible from the interactive level, but seems better than any alternative. `OBEY` files could be used to overcome this problem (inconveniently though).

- `EPSCOL NAME` — Output to a colour pseudo-EPS file.
  Output goes to the `LOCAL` file `NAME`. The format used is described fully in the **DIMFILM** Revived manual and is sufficiently close to 'real' EPS that a very simple Python program running on the emulator host can convert it to that. Either `EPSCOL` or `EPSBIN` must be used if **GPLOT** is run from BATCH, of course. It would be possible to generate real EPS directly in NOS, using 8/12 coding, but that is probably more trouble than it is worth.

[1] See the **GTerm** manual and/or source code for full details.

- EPSBIN NAME — Output to a binary (black or white) pseudo-EPS file.
  While this works fine, it is probably not really useful — EPSCOL is just better in all circumstances.

5 An Example

A simple example of **GPLOT** in use is shown here. The terminal input was:

```
Connected

-----<FF>------------------------------------------------------------
WELCOME TO THE NOS SOFTWARE SYSTEM.
COPYRIGHT CONTROL DATA SYSTEMS INC. 1994.
13/12/30. 16.31.26. TE01
HIGHLANDS COTTAGE COMPUTER CENTER.      NOS 2.8.7 871/871.
FAMILY:
USER NAME: nick
PASSWORD:

JSN: AAAZ, NAMIAF
/attach,abgplot.
/abgplot.
 ===========================================
 GPLOT INTERACTIVE PLOTTING PROGRAM. V0.1
 ===========================================
 BASED ON ULCC DIMFILM BY JOHN GILBERT.
 WRITTEN FOR CDC NOS 2.8 BY NICK GLAZZARD.
 SOME DEVICE & PARSING CODE BY ADRIAN CLARK.
 ===========================================
 ALLOCATED SPACE FOR  1000 DATA POINTS.
 NOTE: OPENING DEVICE, LOADING FONTS WILL TAKE TIME. ALLOW 6 SECS.
 ATTACHED FONT FILE DADIMFO.
? dev gt
? memtest
 ... GOT 1000. 3000. 4000. 5000.
 ... EXPECTED 1000 3000 4000 5000
 OK - DYNAMIC ALLOCATION TEST PASSED.
 GENERATED TEST DATA.
? title "A *lmemory test plot."
? xlab x
? ylab y
? xyl
? ex
    STOP  GPLOT FINISHED.
     3.865 CP SECONDS EXECUTION TIME.
```

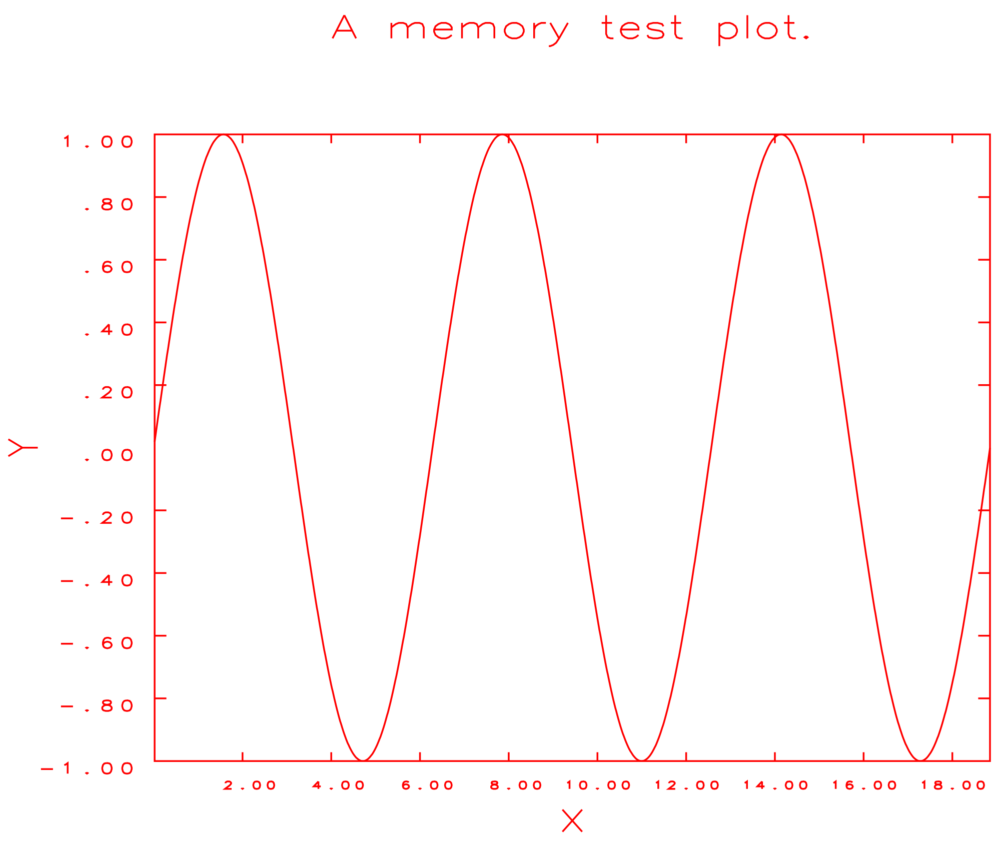

Figure 1: **GPLOT** Output

The result (saved from **GTerm** as SVG) is shown in Figure 1.

6 Building and Maintaining GPLOT on NOS

Both **GPLOT** and the **DIMFILM** library it is built on are maintained entirely under the NOS operating system. The source of **GPLOT** is in a single file, `FTGPLOT`, which is compiled and linked with the **DIMFILM** library, and other components, by a BATCH job in the file `BAGPLOT`. To build **GPLOT**, the following command is used[2]:

```
/SUBMIT,BAGPLOT,NO.
```

Given that all the **DIMFILM** components are available, that is all that is required.

To modify **GPLOT**, edit `FTGPLOT` and rebuild. The FSE editor is a very pleasant full screen editor which makes this process easy.

```
/FSE,FTGPLOT.
```

[2]Instead of `NO`, use `TO` to have the job output go to a file on a wait queue (from which you can get it with `QGET`), or `BC` to have output go to the central line printer.

DIMFILM is a (much) larger piece of software and its source code is held in `MODIFY` format program libraries. `MODIFY` is most like a combination of a macro preprocessor, a source code control system and a patch utility, but there is nothing really like it on modern operating systems. It holds what would be multiple files on a system such as Linux as *named records* in a single file (a *program library*). The named records are called *decks*. There are two types of decks: ones that contain 'ordinary' source code and ones that have much the same purpose as header (include) files. The latter type are called `COMMON` decks. Each deck starts with its name and ends with an end of record, as shown below.

```
MYNAME
  source line 1
  source line 2
  ...
  source line n
(EOR)
```

For a `COMMON` deck:

```
MYCOMN
COMMON
  source line 1
  source line 2
  ...
  source line n
(EOR)
```

As usual in NOS, all names are limited to maximum of 7 characters. Once `MODIFY` has been started, it starts to read *directives* (from the terminal by default when run interactively). Directives begin with an asterisk (*) – at least by default.

Based on the deck names, `MODIFY` commands can be used to make reversible changes to the source code, a bit like a modern source code control system. `MODIFY` is quite complex but it is essential to maintain large scale software (without its 'include' functionality, for example, things would be extremely painful).

The decks in a program library can be listed with the CATALOG command:

```
/GET,PLGRDEV.
/CATALOG,PLGRDEV.
```

Remember to `REWIND` the program library after `CATALOG` otherwise the next `CATALOG` might look like the thing is completely empty! `MODIFY` itself can generate extensive listing of the decks and cross-references between them, as well as which modification sets (used for the source code control aspects) are in effect (or not). This detail is often not needed when `MODIFY` is used as described below.

DIMFILM consists of three program libraries: `PLDIMFM`, which is the portable part of **DIMFILM**, including in this version, much of the output device specific code; `PLGRDEV`, which provides low level support for the graphics devices; and `PLUTILS` which contains code useful to many programs (such as text file parsing functions).

Each of the program libraries has a BATCH job file which builds an ‘object library’ (although this is not the NOS terminology) from the source. **GPLOT** then ‘links’ (also not NOS terminology) with these binary libraries to produce a binary that can be loaded (and hence run) without further external requirements.

Each of the three ‘object libraries’ can be built independently of each other and hence in any order. The relationships between the key source files are shown in Table 1.

Source	Build Job	Result	Purpose
PLDIMFM	BADIMFM	LBDIMFM	**DIMFILM** library
PLGRDEV	BAGRDEV	LBGRDEV	Graphics device library
PLUTILS	BAUTILS	LBUTILS	Utility routine library
FTGPLOT	BAGPLOT	ABGPLOT	**GPLOT** executable

Table 1: Files used to build **GPLOT** and **DIMFILM**

Maintaining the source for the libraries is not so straightforward as it is for **GPLOT** itself, as it must be done through `MODIFY`. It isn’t appropriate to go into great detail here, but some useful ‘recipes’ (to which you need to make the ‘obvious’ interpolations) are:

```
/MODIFY,P=OPLNAME,L=0,C=0,S=SRCOUT.
*EDIT DECKNAM
*DECK DECKNAM
(empty line terminates directive input)
```

This extracts the deck called `DECKNAM` to the file `SRCOUT`. All the source in the program library could be extracted by adding `,F.` to the command and omitting the directives. The program library could be recreated from this ‘full’ `SRCOUT` file.

```
/MODIFY,P=PLORG,N=PLTEMP,L=0,C=0,F.
*CREATE INFILE
(empty line terminates directive input)
/REPLACE,PLTEMP=PLORG.
```

This adds new decks defined in `INFILE`, or replaces in their entirety existing decks, if the decks in `INFILE` have the same names as existing ones. The simplest way of working with `MODIFY`, using FSE, is to extract a deck to be changed (akin to a source or include file) as source using the first recipe, edit the extracted source file with FSE, then replace it with the second recipe.

To generate input for a compiler (i.e. source with all the ‘include files’ in COMMON decks included inline, etc.) use:

```
/MODIFY,P=PLORG,C,L,F.
/FTN5,I=COMPILE.
```

`L=0` turns off `MODIFY` listing output and a specific name can be given for the compilable output with `C=COMPNAM` (otherwise output goes to a file called `COMPILE`).

There are many other commands and directives for doing source code control and patch/edit tasks, but it isn’t clear how useful these are when a powerful full screen editor (FSE) is available. No doubt they can be useful, but perhaps they are not necessary for this project.

6.1 Step-by-step Installation Guide

You need to be running NOS 2.8 (I have only tested with 2.8.7) on dtCYBER with the FTN5 compiler. You need to install from an account which can use one magnetic tape device. You need to be able to submit batch jobs from the card reader.

The source and working binaries for **GPLOT** are to be found in a compressed tar file: `gplotnos.tgz` This contains the following items:

- `gplot.tap` : This is a tap format tape image containing a multi-file set (see Section 6-14 of NOS Version 2 Reference Set, Volume 2 – Guide to System Usage – 60459670). Both the binaries built for NOS 2.8.7 and all source code needed to build and run **DIMFILM** and **GPLOT** are on this tape.
- `bardgp.txt` : This is a batch job to be submitted from the emulator host to the card reader. This reads `gplot.tap` and creates **GPLOT**'s files as permanent files in the account of the user specified in the batch job. The USER card will need to be edited appropriately before the job is submitted.
- `gplot.pdf` : This manual in PDF format.
- `gplotmanual.tgz` : Source for the manual.
- `toeps.py` : Python program to convert plots output with the EPSCOL or EPSBIN devices to one or more Encapsulated PostScript files on the host. See Section 6.3 for an example of how to use this.

Proceed as follows:

1. On the emulator host, unarchive `gplotnos.tgz` in some convenient location.

```
        $ tar xvzf gplotnos.tgz
        $ cd gplot
```

2. Copy `gplot.tap` to the usual location from which you access tape images.
3. Copy `bardgp.txt` to the usual location from which you submit batch jobs to the card reader.
4. Submit `bardgp.txt` to the card reader using the dtCYBER operator interface (or your usual method). E.g.:

```
        Operator> lc 12,4,cards/bardgp.txt
```

   The 'installation' batch job will start.

5. Observe the dtCYBER console. When prompted to, switch to the `RESOURCE REQUESTS` display with:

```
        E,P.
```

   You should see a request to mount `VSN GPLT15` appear.

File	Purpose
PLDIMFM	MODIFY program library for **DIMFILM**
BADIMFM	Batch job to build the binary **DIMFILM** library, LBDIMFM
LBDIMFM	Pre-built binary **DIMFILM** library for NOS 2.8.7 on a CYBER 175
PLGRDEV	MODIFY program library for the graphics device library
BAGRDEV	Batch job to build the binary graphics device library, LBGRDEV
LBGRDEV	Pre-built binary graphics device library for NOS 2.8.7 on a CYBER 175
PLUTILS	MODIFY program library for utility routines
BAUTILS	Batch job to build the binary utility library, LBUTILS
LBUTILS	Pre-built binary utility routine library for NOS 2.8.7 on a CYBER 175
FTGPLOT	Source code for the **GPLOT** program
BAGPLOT	Batch job to compile and link **GPLOT** to create ABGPLOT
ABGPLOT	Pre-built binary **GPLOT** executable for NOS 2.8.7 on a CYBER 175
DADIMFO	**DIMFILM** font definitions data file
BAWRTGP	Batch job to write a new **GPLOT** install tape

Table 2: Files installed by `bardgp.txt`

6. Using the dtCYBER operator interface, load `gplot.tap` on a suitable tape drive. For example:

```
        Operator> lt 13,0,1,r,tapes/gplot.tap
```

   The request in the `E,P.` display should go away after a few seconds. (At least, that is what happens on my system. If not try the console command: `VSN,51,GPLT15.` adjusting 51 to be the correct EST ordinal for the tape drive as needed. I don't think this will be required, though.)

7. Optionally switch to the `E,T.` console display (`TAPE STATUS`) to see the progress of copying files to disk.

8. When the batch job completes (hopefully successfully), the permanent files shown in Table 2 will appear in the catalog of the `USER` specified in the batch job.

9. If the machine running NOS 2.8.7 is a CYBER 175, it should be possible to use the **GPLOT** program binary, `ABGPLOT`, immediately. Otherwise, it may be necessary to rebuild from source. See Section 6.2 for instructions.

6.2 Rebuilding from source

Once **GPLOT** has been installed by the `bardgp.txt` batch job, it should be possible to rebuild the binaries from an interactive session using the following commands:

```
/ GET,BADIMFM.
/ SUBMIT,BADIMFM,NO.
/ GET,BAGRDEV.
/ SUBMIT,BAGRDEV,NO.
/ GET,BAUTILS.
/ SUBMIT,BAUTILS,NO.
/ GET,BAGPLOT.
/ SUBMIT,BAGPLOT,NO.
```

The binary libraries created by the first three SUBMITs must be built completely before the final SUBMIT. I'm not sure this is guaranteed to happen if you don't manually wait for it! The first three SUBMITs are independent of one another, so issuing them one after another without waiting should be fine.

6.3 Using EPSCOL and EPSBIN devices

To create a plot file with the EPSCOL or EPSBIN devices and send it to the emulator host file system, you can use the following example as a guide (there are alternatives using magnetic tapes).

Create the plot file on NOS:

```
/attach,abgplot.
/abgplot.
 ===========================================
 GPLOT INTERACTIVE PLOTTING PROGRAM. V0.1
 ===========================================
 BASED ON ULCC DIMFILM BY JOHN GILBERT.
 WRITTEN FOR CDC NOS 2.8 BY NICK GLAZZARD.
 SOME DEVICE & PARSING CODE BY ADRIAN CLARK.
 ===========================================
 ALLOCATED SPACE FOR  1000 DATA POINTS.
 NOTE: OPENING DEVICE, LOADING FONTS WILL TAKE TIME. ALLOW 6 SECS.
 ATTACHED FONT FILE DADIMFO.
? dev epscol opeps
? memtest
 ... GOT 1000. 3000. 4000. 5000.
 ... EXPECTED 1000 3000 4000 5000
 OK - DYNAMIC ALLOCATION TEST PASSED.
 GENERATED TEST DATA.
? xyl
? ex
    STOP  GPLOT FINISHED.
      3.611 CP SECONDS EXECUTION TIME.
/enquire,f.

 LOCAL FILE INFORMATION.

  FILENAME  LENGTH/PRUS  TYPE  STATUS     FS  LEVEL
   ABGPLOT       1078     PM.*  EOR
   DADIMFO        511     PM.*  I/C
   INPUT                  TT.
   INPUT*           1     IN.*  EOR        NAD
   OPEPS          190     LO.   EOR WRITE
   OUTPUT                 TT.
   ZZZZZOU                TT.

  TOTAL = 8
/route,opeps,dc=pu.
 ROUTE COMPLETE.  JSN IS AABI.
```

This will send the plot file to the card punch, whereupon it should appear in a folder on the emulator host. The details may vary (I run a slightly modified version of dtCYBER which automates some tasks in conjunction with a helper program that watches folders) but on my machine the punched output will appear in a file such as:

```
CP3446_20150326_155617.pun
```

You may have to explicitly unload the card reader on your dtCYBER version – I'm not sure. In any case, once a card punch output file has been retrieved, it can be converted to EPS files using:

```
toeps.py CP3446_20150326_155617.pun
```

(Assuming `toeps.py` has been made executable and is on your path, etc.)

A separate EPS file will be created for each 'frame' in the **GPLOT** plot file. The original name of the local file to which output was sent in **GPLOT** is stored in this file and is used as the basis for the EPS output file names. In this case:

```
opeps001.eps
```

will be created. If there were multiple 'frames' in the plot file, these would be named `opeps001.eps`, `opeps002.eps`, etc.

7 GPLOT and DIMFILM for Unix systems

GPLOT and **DIMFILM** are maintained under CDC NOS 2.8. There is a port from NOS to Mac OS X Unix, though, which will probably also work on Linux.

There is a (rather obvious) prerequisite: `gfortran` must be installed to build *or run* Unix **GPLOT**. It should be possible to link **GPLOT** statically, but I have not done that.

The port started from the following files, transferred from the CDC CYBER mainframe:

- `PLDIMFM.TXT` : **DIMFILM** source deck.
- `PLGRDEV.TXT` : Graphics device source deck.
- `PLUTILS.TXT` : Utility library source deck.
- `FTGPLOT.TXT` : **GPLOT** Fortran source code.

These files were created by these NOS commands:

```
modify,p=pldimfm,l=0,c=0,s=srdimfm,f.
route,srdimfm,dc=pu.
...
route,ftgplot,dc=pu.
```

(Repeated for the `PLGRDEV` and `PLUTILS` MODIFY source libraries in the obvious way.)

The output to the card punch folder on the mainframe simulator host was then transferred to a `unix` subdirectory. The card punch output files were renamed by hand to get the desired file names.

The contents of the MODIFY source decks were then split in to separate files by a simple Python program:

```
python modsplit.py -u -s builddim.sh -l dimfilm --omit "getbyt" PLDIMFM.TXT
```

which put the files in to a folder: `dimfilm` and accumulated a simple build script in builddim.sh. In this case, it omitted module `getbyt` which contains CDC 6600 assembler code.

The source of **GPLOT** itself was created from `FTGPLOT.TXT` with the aid of upcase.py (I like my Fortran source in upper case!), yielding `ftgplot.f`.

The source files were then modified to the minimal extent possible to work on Unix. This involved removing the Common Memory Manager based dynamic data array allocation, and implementing a NOS style command line parsing function and a minimal version of the `PF()` subroutine. In fact, very little needed to be changed.

To rebuild the **DIMFILM**, device and utility libraries, and then the **GPLOT** program, use:

```
sh buildall.sh
```

On completion, the program: `gplot` should have been created. This can be run pretty much as per the CDC NOS version. For example:

```
./gplot
./gplot obey=obhere,parm=gt
./gplot obey=obhere,parm="epscol plot1"
python ../gitprojects/utility/toeps.py PLOT1
```

and so on. Note that all files created by **GPLOT** have `UPPER CASE` names and `OBEY` file names on the command line must be no more than 7 characters long (and without any extension)! Note that you will need to escape some command line characters used by **GPLOT** in the usual way for the shell you are using. In particular, `$` will need escaping.

The file : `DADIMFO` contains the font definitions and must be present.

The Unix version functions as much like the NOS version as possible — even if Unix specific modifications, such as directly generating EPS files would be "obviously better". The current goal of the Unix version is just to help keep **DIMFILM** alive by having it run on a platform that more than a handful of people have access to.

The source of this documentation can be found in the `doc` folder. To create a PDF file from the LaTeX source, use:

```
xelatex gplot.tex
```

Source for the **DIMFILM** manual is not currently available (only the PDF). That source isn't in a very satisfactory state at the moment.