Using S-Lang Scripts for Chandra Data Analysis

I present some simple S-Lang scripts for Chandra data analysis. The examples are meant to illustrate:

- how to set up a script;
- how to get at CIAO S-Lang modules;
- how to deal with script arguments (one way);
- how to execute commands from within a script;
- how to plot, using CHIPS
- how to read simple FITS files.

The examples include:

- HELLO.SL - Just to make sure we're all starting on the same page;
- PLDTF.SL – Read HRC dtf1 file, fix dead time factors and plot, using CHIPS;
- GETCOUNTS.SL – Get counts in regions, using DMEXTRACT; designed to be launched from DS9, but may also be run from command line;

All scripts and their supporting subroutines may be found in /home/fap/bin. To run, the CIAO environment must be set up.

```
#!/usr/bin/env /soft/ciao/ots/slang.v1.4.4/slsh/slsh

_auto_declare =1;
import("chips"); % ensure the Varmm functions are available
() = printf("Hello, World\n");
```

Usage:

```
calf-3: ciao23
 CIAO configuration is complete...
 CIAO version     :  CIAO 2.3 Thursday, October 31, 2002
 Proposal Toolkit version :  NRA 4.0 Saturday, October 13, 2001
 bin dir          :  /soft/ciao2.3/bin
calf-4: hello.sl
Hello, World
calf-5:
```

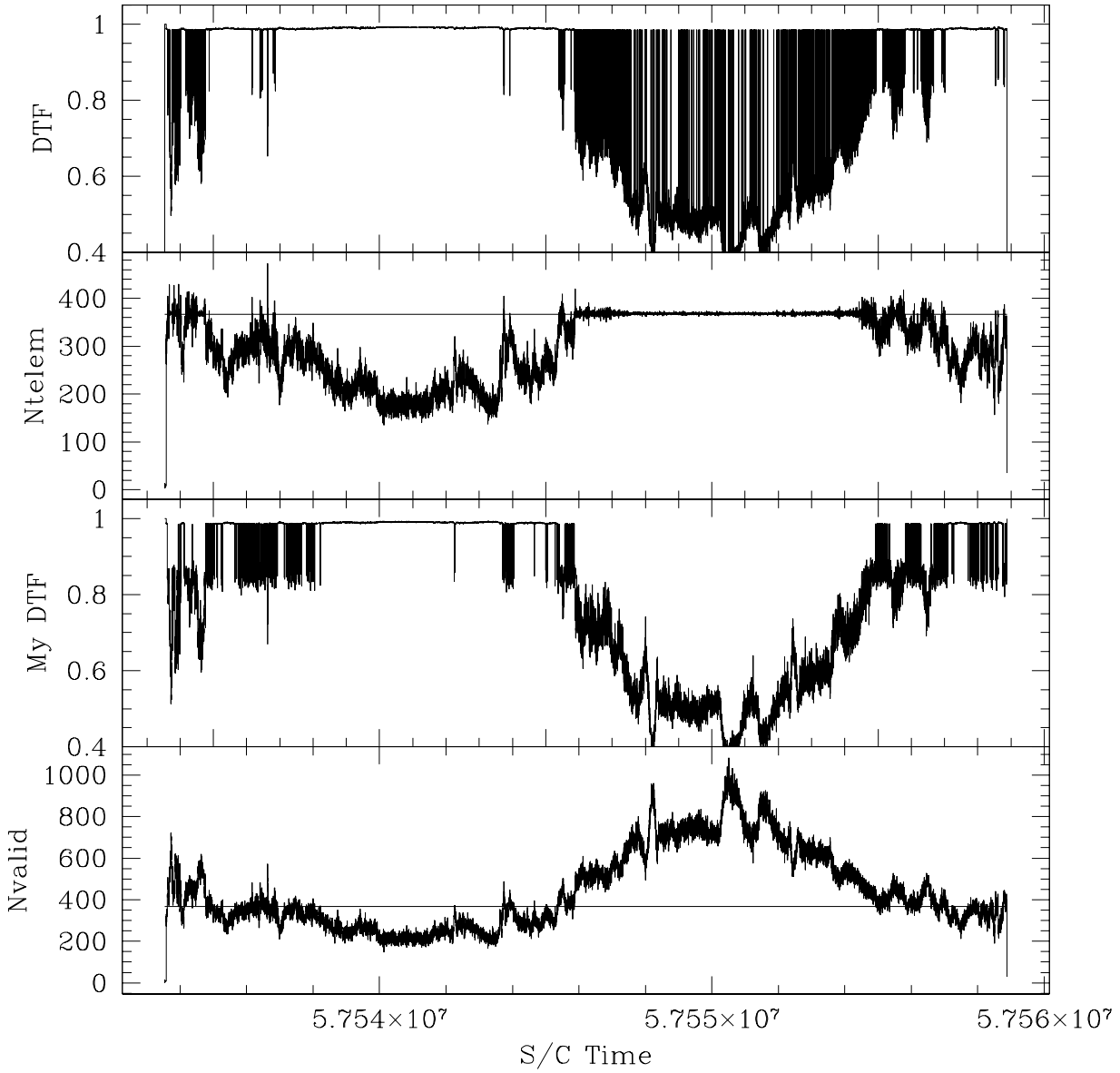
Displaying and correcting HRC dead time corrections:

The HRC dead time correction is estimated in 2 second intervals, and depends on the total number of events occurring in the interval, the number that pass the hardware checks, and the actual number telemetered. These quantities are contained in the FITS binary table file HRC*dtf1.fits.

The S-Lang script pldtf.sl displays the dead time corrections and attempts to fix them during times of telemetry saturation.

Usage:

pldtf.sl hrcf...dtf1.fits


DTF
1
0.8
0.6
0.4
Ntelem
400
300
200
100
0
My DTF
1
0.8
0.6
0.4
Nvalid
1000
800
600
400
200
0
5.754×10⁷
5.755×10⁷
5.756×10⁷
S/C Time

```
#!/usr/bin/env /soft/ciao2.3/ots/slang.v1.4.4/slsh/slsh

%
% 1/16/2003
% fap
%

%
% Plot data from Table 1 of Mike and Adam's HRC Deadtime Memo
%

%
% Usage:
%
% pldtf.sl <input dtf file>
%

_auto_declare =1;
import("chips");
()=evalfile("/home/fap/bin/myfuns.sl");        % Load personal library of S-Lang functions

infile  = __argv[1];                           % __argv array contains arguments to script, as strings.
                                               % __argc contains the number of arguments
%
% Read in entire table 1
%

dtfcols = readfile(infile);                    % readfile is a chips module routine to read a FITS binary table
                                               % into a structure, containing arrays for each column
%
% Get various data arrays
```

```
%

Time = dtfcols.TIME;
DTF  = dtfcols.DTF;
DTFERR = dtfcols.DTF_ERR;
Ntelem = dtfcols.PROC_EVT_COUNT;
Ntot = dtfcols.TOTAL_EVT_COUNT;
Nvalid = dtfcols.VALID_EVT_COUNT;

%
% Guard against division by zero
%

Nvalid[where(Nvalid==0)]=1;                    % Note vector operation and the use of the where statement
                                               % to specify appropriate array indices

%
% Determine TM saturation times
%

sattime = where(Nvalid>367);
unsattime = where(Nvalid<=367);

dt = Double_Type[length(DTF)];
dt[unsattime] = 19.5e-6*Ntot[unsattime]*(1.0-((Nvalid[unsattime]-Ntelem[unsattime])/Nvalid[unsattime]))+49.5e-
6*Ntelem[unsattime];
dt[sattime] = 2.0*(Nvalid[sattime]-Ntelem[sattime])/Nvalid[sattime];

dt = 1.0-dt/2.0;

() = chips_eval("redraw off");
() = chips_eval("split 4");
```

```
() = chips_eval("d 1");
()=curve(Time,DTF);
() = chips_eval("d 1 c 1 step");
() = chips_eval("d 1 c 1 symbol none");
() = chips_eval("d 1 tickvals x off");
() = chips_eval("d 1 ylabel \"DTF\"");
() = chips_eval("d 1 limits y 0.4 1.05");
() = chips_eval("redraw");

() = chips_eval("d 2");
()=curve(Time,Ntelem);
() = chips_eval("d 2 c 1 step");
() = chips_eval("d 2 c 1 symbol none");
() = chips_eval("d 2 tickvals x off");
() = chips_eval("d 2 ylabel \"Ntelem\"");
() = chips_eval(sprintf("d 2 line %e %f %e %f",Time[0],367.0,Time[length(Time)-1],367.0));
() = chips_eval("redraw");

() = chips_eval("d 3");
() = chips_eval("d 3 limits y 0.4 1.05");
()=curve(Time,dt);
() = chips_eval("d 3 c 1 step");
() = chips_eval("d 3 c 1 symbol none");
() = chips_eval("d 3 tickvals x off");
() = chips_eval("d 3 ylabel \"My DTF\"");
() = chips_eval("redraw");

() = chips_eval("d 4");
()=curve(Time,Nvalid);
() = chips_eval("d 4 c 1 step");
() = chips_eval("d 4 c 1 symbol none");
```

```
() = chips_eval("d 4 ylabel \"Nvalid\"");
() = chips_eval("d 4 xlabel \"S/C Time\"");
() = chips_eval(sprintf("d 4 line %e %f %e %f",Time[0],367.0,Time[length(Time)-1],367.0));
() = chips_eval("redraw");

() = chips_eval("print postfile /tmp/pldtf.ps");

docmd("gv /tmp/pldtf.ps&");
```

The file myfuns.sl contains the following:

```
%
% Collect here some useful functions of my own, inspired by things I saw
% in other s-lang routines.
%
% fap 12/19/2002
%

% docmd just executes a command after printing a message out

define docmd(cmdstr)
{
        () = printf("Executing %s...\n",cmdstr);
        () = system(cmdstr);
}

% doproc executes a command like docmd, but collects the output into
% a string array and returns that.

define doproc(cmdstr)
{
        variable pp, cmdout, errstr;
        pp = popen(cmdstr,"r");
        cmdout = fgetslines(pp);
        cmdout = array_map(String_Type,&strtrim,cmdout);
        if(cmdout != NULL){
          return cmdout;
        }
        else{
```

```
            errstr = sprintf("Could not read process %s",cmdstr);
            error( "\n** ERROR **\n " + errstr + "\n" );
            exit(1);
        }
        () = pclose(pp);
}

% This is just a more user-friendly error message

define myerror (estr) {
    error( "\n** ERROR **\n " + estr + "\n" );
}
```

Getting Counts in Regions:

The S-Lang script getcounts.sl determines counts in regions using dmextract. It may be executed from the command line or launched from ds9.

Usage:

```
calf-14: getcounts.sl 1838evt2.fits.gz "circle(4096,4096,20);"
================================================================

Getting Counts in Regions

Fri Jan 31 08:39:14 2003

Input File: 1838evt2.fits.gz

Source Region(s) Written to /tmp/src.reg:

circle(4096,4096,20)

No Background Regions Have Been Specified.

Executing Command:
dmextract infile=1838evt2.fits.gz"[bin sky=@/tmp/src.reg]" outfile=/tmp/dme.fits clobber=yes

Source Region       Counts    Area
-------------------       ------       ----
circle(4096,4096,20)              55.00      1256.64
calf-15:
```

```
#!/usr/bin/env /soft/ciao/ots/slang.v1.4.4/slsh/slsh

_auto_declare =1;
import("chips"); % ensure the Varmm functions are available

message("================================================================================");
message("");
message("Getting Counts in Regions");
message("");
td=time();
message(td);
message("");

% Check argument list for source and background regions

switch(__argc)      % Simple Argument Checking

        {case 1: % Should never come here unless running from command line with no file specified
                message("Error: No Input File or Source Regions Specified");
                exit(1);
        }

        {case 2: % Data file specified, but no regions
                message("Error: No Source Regions Specified");
                exit(1);
        }

        {case 3: % Data file and source region(s) specified
                infile = __argv[1];
                sreg   = __argv[2];
```

```
line = sprintf("Input File:\t%s\n",infile);
message(line);

message("Source Region(s) Written to /tmp/src.reg:");
message("");
()=system("rm -f /tmp/src.reg");
regions=strchop(sreg,';',0);       % Split string into array based on ";" delimiter
fp=fopen("/tmp/src.reg","w");
for(i=0;i<length(regions)-1;i++){
        nc=fprintf(fp,"%s\n",regions[i]);
        message(regions[i]);
}
()=fclose(fp);

message("");
message("No Background Regions Have Been Specified.");
message("");
cmd = sprintf("dmextract infile=%s\"[bin sky=@/tmp/src.reg]\" outfile=/tmp/dme.fits
clobber=yes",infile);
message("Executing Command:");
message(cmd);
() = system(cmd);

dme = readbintab("/tmp/dme.fits");

message("");

lcounts = length(dme.COUNTS);
maxreg = 0;
for(i=0;i<lcounts;i++){
```

```
                        if(strlen(regions[i])>maxreg){maxreg=strlen(regions[i]);}
                }


                line = sprintf("%-*s\t%s\t%s",maxreg,"Source Region","Counts","Area");
                message(line);
                minuses="-----------------------------------------------------------------------------------------------";
                line = sprintf("%.*s\t------\t----",maxreg,minuses);
                message(line);

                for(i=0;i<lcounts;i++){
                        counts = dme.COUNTS[i];
                        area   = dme.AREA[i];
                        line = sprintf("%-
*s\t%.2f\t%.2f",maxreg,regions[i],dme.COUNTS[i],dme.AREA[i]);
                        message(line);
                }

        }

        {case 4: % Data file, source and bgd region(s) specified
                infile = __argv[1];
                sreg   = __argv[2];
                breg   = __argv[3];

                line = sprintf("Input File:\t%s\n",infile);
                message(line);

                message("Source Region(s) Written to /tmp/src.reg:");
                message("");
                ()=system("rm -f /tmp/src.reg");
```

```
regions=strchop(sreg,';',0);
fp=fopen("/tmp/src.reg","w");
for(i=0;i<length(regions)-1;i++){
        nc=fprintf(fp,"%s\n",regions[i]);
        message(regions[i]);
}
()=fclose(fp);

message("");

message("Background Region(s) Written to /tmp/bgd.reg:");
message("");
()=system("rm -f /tmp/bgd.reg");
bregions=strchop(breg,';',0);
fp=fopen("/tmp/bgd.reg","w");
for(i=0;i<length(bregions)-1;i++){
        nc=fprintf(fp,"%s\n",bregions[i]);
        message(bregions[i]);
}
()=fclose(fp);

message("");

cmd = sprintf("dmextract infile=%s\"[bin sky=@/tmp/src.reg]\" outfile=/tmp/dme.fits
bkg=%s\"[bin sky=@/tmp/bgd.reg]\" clobber=yes",infile,infile);
message("Executing Command:");
message(cmd);
() = system(cmd);

dme = readbintab("/tmp/dme.fits");
```

```
                message("");

                lcounts = length(dme.COUNTS);
                maxreg = 0;
                for(i=0;i<lcounts;i++){
                        if(strlen(regions[i])>maxreg){maxreg=strlen(regions[i]);}
                }

                line = sprintf("%-*s\t%s\t%s\t%s\t%s\t%s",maxreg,"Source
Region","Counts","Area","Bg Cts","Bg Area","Net Cts","Net Err");
                message(line);
                minuses="-----------------------------------------------------------------------------------------------";
                line = sprintf("%.*s\t------\t----\t------\t-------\t-------\t-------",maxreg,minuses);
                message(line);

                for(i=0;i<lcounts;i++){
                        line =
sprintf("%-
*s\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f",maxreg,regions[i],dme.COUNTS[i],dme.AREA[i],dme.BG_COUNTS[i],d
me.BG_AREA[i],dme.NET_COUNTS[i],dme.NET_ERR[i]);
                        message(line);
                }

        }
```

The script getcounts.sl may be launched directly from ds9, by loading the ds9 “analysis file” /home/fap/bin/ciao.ds9, listed here. The script will then compute counts in the regions displayed in ds9 and output the results in a pop-up text window.

```
# Some Simple CIAO Tasks Executed from DS9
#---
#hmenu CIAO

Get Counts in Regions
*.fits *.fits.gz
menu
/home/fap/bin/getcounts.sl $filename $regions(source,ciao) $regions(background,ciao) | $text

Radial Profile
*.fits *.fits.gz
menu
/home/fap/bin/rprof.sl $filename $regions(source,ds9) $regions(background,ds9)

param sspar
var1 entry {Aspect Solution File:} asol1.fits {pcad...asol1.fits}
var2 menu {Model Spectrum:} PL|TB|BB {power law, bremss, black body}
var3 entry {Output Filename Root:} test {output files start with...}
end

param psepar
@/soft/ciao/param/psextract.par
end

Simple Spectral Fit
*.fits *.fits.gz
```

```
menu
$param(sspar);/home/fap/bin/simplespec.sl $var1 $var2 $var3 |$text

Testing...
*.fits *.fits.gz
menu
$param(psepar);/home/fap/bin/test.sl $var1 $var2 $var3
#end
```