CIAO

CIAO 2.2 & S-Lang

What is S-Lang?

- A popular interpreted language (so no compile/link stages) created by John Davis of the Center for Space Research at MIT.
- Used by a number of diverse programs (eg *jed*, *mutt*, *ISIS* & *CIAO*) to provide a scripting/control language.
- Has a C-like syntax without the worries of memory management.
- Provides a powerful array-based numerical computing environment.
- Programs can easily add new commands to extend the language.

For more information see

http://www.s-lang.org/

cxc.harvard.edu/ciao/ahelp/slang.html

Our First S-Lang function

```
define hello() { print("Hello World!"); }
```

Where is S-Lang?

- It is available from Sherpa and ChIPS.

```
chips> define hw() { print("Hello World!"); }
chips> hw()
Hello World!
chips>
```

```
ahelp slang-ciao
ahelp slang-chips
ahelp slang-sherpa
```

- It is also possible to "load" ChIPS into a S-Lang script. This allows a program such as *jed* to use ChIPS to create plots. A more useful application is the creation of S-Lang "scripts" similar to - but more powerful than - the shell scripts we currently provide. Coming soon to a CIAO website near you!

- CIAO contains the Varmm (Variable, Math, and Macro) library, which extends S-Lang with useful routines for the Astronomer (eg FITS in/output, simple math operations).

```
ahelp varmm
```

Why bother?

- Sherpa and ChIPS both have their own command languages (ahelp sherpa and ahelp chips) but they are designed for *interactive* use. Neither allow you to manipulate data, or provide loops and conditional statements.
- Although "command scripts" can be used to automate tasks, what is needed is a powerful and versatile language that can be used within ChIPS and Sherpa.
- S-Lang was chosen since it is small, designed to be "embedded" within applications, and has a powerful array-manipulation syntax which simplifies many mathematical operations.
- GUIDE is written in S-Lang.
- A number of S-Lang routines are available from the CIAO website (in the Download Scripts section) that illustrate some of the possibilities of S-Lang within CIAO.

cxc.harvard.edu/ciao/download_scripts.html

A source of confusion

ChIPS & Sherpa parse each line of input and work out whether to handle it themselves or to pass it on to the S-Lang interpreter. This has good and bad points:

- "good"
  You do not need to pre-declare variables, finish with a semi-colon, or worry about the stack.

- "bad"
  Statements can not span multiple lines, error messages are often complex, and it is possible - if you try hard enough - to confuse the parser.

This scheme means that ChIPS and Sherpa command scripts can contain S-Lang statements. However, the S-Lang interpreter does not understand ChIPS or Sherpa commands. Instead, "pure" S-Lang code has to use the chips_eval() and sherpa_eval() commands, which sends the arguments to the ChIPS and Sherpa interpreters respectively.

Loading/running S-Lang code

```
unix% chips foo.chp
unix% chips --batch foo.chp
```

The file foo.chp can contain ChIPS and S-Lang commands (those that fit onto 1 line). The '--batch' option exits ChIPS after running through the commands, rather than leaving you at the prompt.

```
unix% chips --slscript foo.sl
unix% chips --batch --slscript foo.sl
```

The file foo.sl can contain S-Lang commands (including multi-line statements, but variables must be pe-declared and trailing semi-colons must be included).

```
chips> () = evalfile("foo.sl");
chips> evalfile("foo.sl")
1
```

S-Lang code can also be loaded into a running ChIPS or Sherpa session with the evalfile() command. The evalfile() command returns a 1 if succesfull: in the first instance we ignore the value (the '()=...' syntax), and in the second we let ChIPS clear the return value from the stack (the cleared values get printed to the screen).

A digression about calculators

S-Lang passes function arguments and return values by adding/removing them from the "stack" (essentially a region of memory which remembers the order that items were placed into it). It is therefore important to use the correct number of function arguments, and the correct number of return values in S-Lang code - otherwise the code will stop with "Stack underflow/overflow" errors.

However, both ChIPS and Sherpa will clean up the stack after processing a S-Lang command, and print out any remaining items (note that this is only true from the chips> or sherpa> prompts). This makes using S-Lang easier, and turns ChIPS and Sherpa into calculators:

```
chips> 12.0/5
2.4
chips> sin(PI/4)
0.707107
```

although sherpa can get confused

```
sherpa> sin(PI/4)
Error: Model Error has been detected.
  --   Parameter type mismatch.
```

Example - plotting a pha file

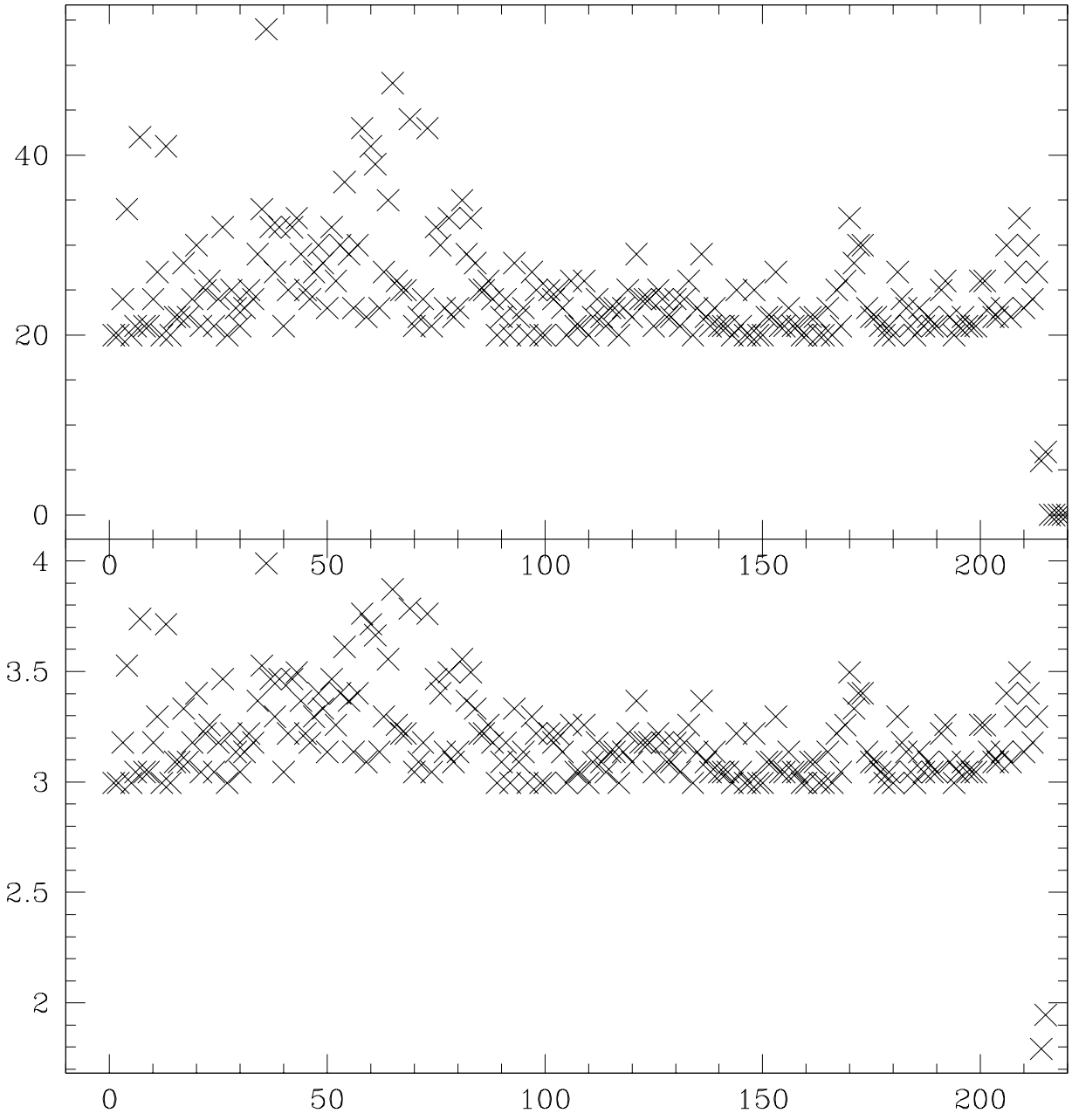
/data/ciao_demo/threads/S-Lang/plot_pha/

Directly from ChIPS:

```
chips> variable pha = readpha( "source.pi" );
chips> print(pha)
_filename         =  source.pi
_path             =
    /data/ciao_demo/threads/S-Lang/plot_pha/
_filter           =  NULL
_header           =  String_Type[428]
backscal          =  0.00302159
areascal          =  1
exptime           =  51164.2
_ncols            =  11
_nrows            =  1024
channels          =  Float_Type[554]
counts            =  Float_Type[554]
grouping          =  Integer_Type[1024]
qualityflags      =  Integer_Type[554]
phachans          =  Integer_Type[554]
errors            =  Float_Type[554]
background        =  none
arf               =  none
response          =  none
numgroups         =  554
numchans          =  1024
chips> () = chips_eval("clear");
chips> () = chips_eval("split 2");
chips> () = curve( pha.channels, pha.counts );
chips> () = chips_eval("limits x -10 220");
chips> () = chips_eval("d 2 limits x -10 220");
chips> () = curve( pha.channels, log(pha.counts) );
```

The file plot_pha.scp contains these commands and can be executed using:

```
unix% chips plot_pha.scp
```


40
20
0
0
50
100
150
200
4
3.5
3
2.5
2
0
50
100
150
200

These commands could be written as a function (here called plot_pha1):

```
define plot_pha1 (phaname) {
    variable pha = readpha( phaname );
    if ( pha == NULL )
        error("Unable to read pha file " + phaname);

    () = chips_eval("redraw off");
    () = chips_eval("clear");
    () = chips_eval("split 2");
    () = curve( pha.channels, pha.counts );
    () = chips_eval("limits x -10 220");

    % second plot
    () = chips_eval("d 2 limits x -10 220");
    () = curve( pha.channels, log(pha.counts) );

    () = chips_eval("redraw on");

}
```

If the file plot_pha1.sl contains the above, then it can be loaded and called as shown below:

```
chips> () = evalfile("plot_pha1.sl")
chips> plot_pha1( "source.pi" )
```

The following ahelp pages discuss some of the commands and concepts used here:

```
ahelp readfile
ahelp -c slang print
ahelp chips_eval
ahelp slang-ciao
```

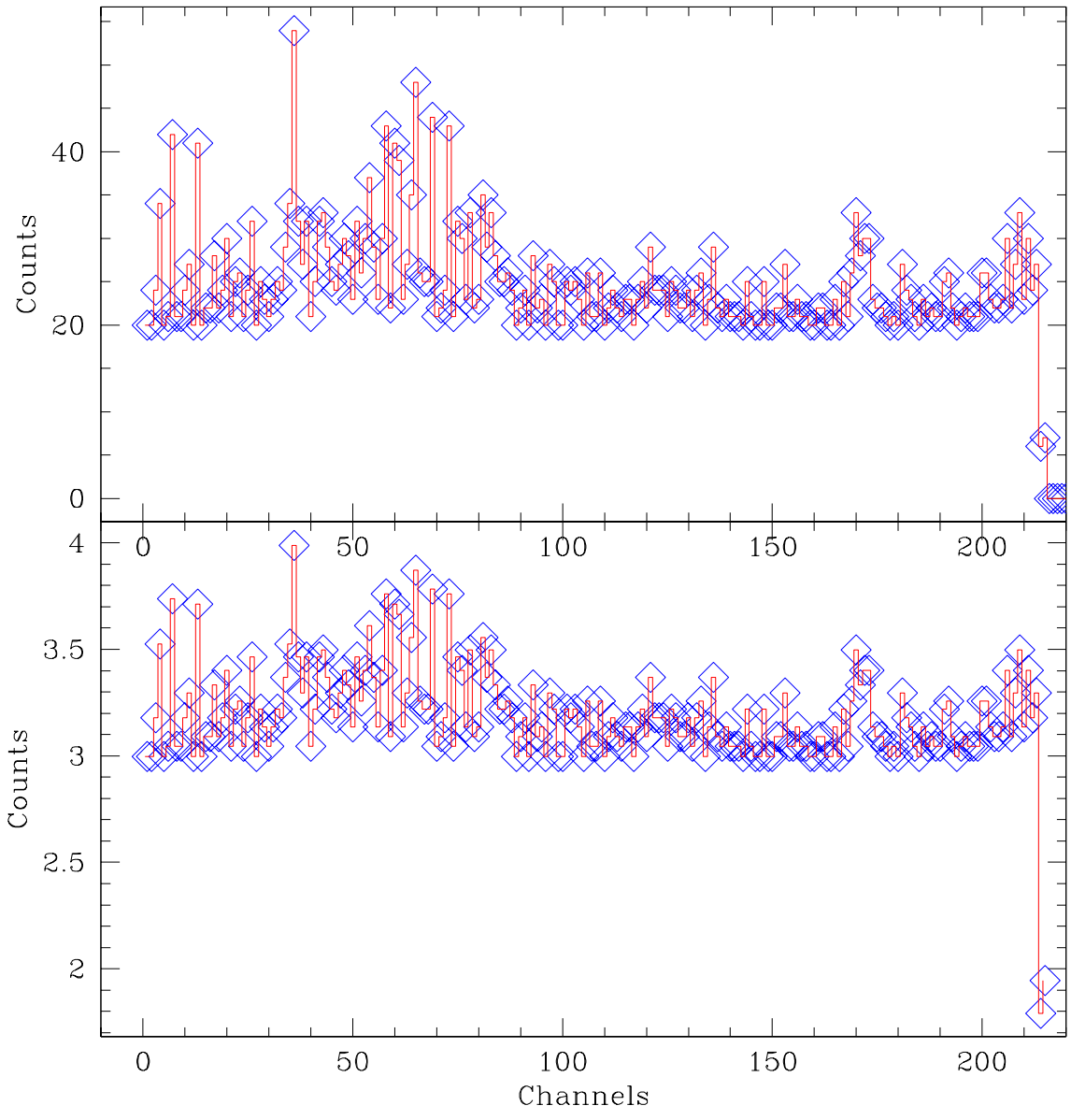
CIAO

Changing the plot

In the previous example, the plot was created using chips_eval() to call ChIPS commands (other than the curve() command). The most common plot attributes (eg line color, symbol style) can also be changed using the chips state object (ahelp slang-chips) as shown in plot_pha2.sl:

```
% label_axes( xtext, ytext )
% - makes labelling axes easier
%
define label_axes( xl, yl ) {
    () = chips_eval("xlabel '" + xl + "'");
    () = chips_eval("ylabel '" + yl + "'");
}


define plot_pha2 (phaname) {
    variable pha = readpha( phaname );
    if ( pha == NULL )
        error("Unable to read pha file " + phaname);

    % store the current chips settings
    variable oldchips = @chips;

    % set plot styles
    chips.symbolstyle = _chips->diamond;
    chips.symbolcolor = _chips->blue;
    chips.curvestyle  = _chips->histo;
    chips.curvecolor  = _chips->red;

    % first plot
    () = chips_eval("redraw off");
    () = chips_eval("clear");
    () = chips_eval("split 2");
    () = curve( pha.channels, pha.counts );
    () = chips_eval("limits x -10 220");
    label_axes( "", "Counts" );
```

```
        % second plot
        () = chips_eval("d 2 limits x -10 220");
        () = curve( pha.channels, log(pha.counts) );
        label_axes( "Channels", "Counts" );

        () = chips_eval("redraw on");

        % restore the chips plotting styles
        set_state( "chips", oldchips );
}
```



CIAO

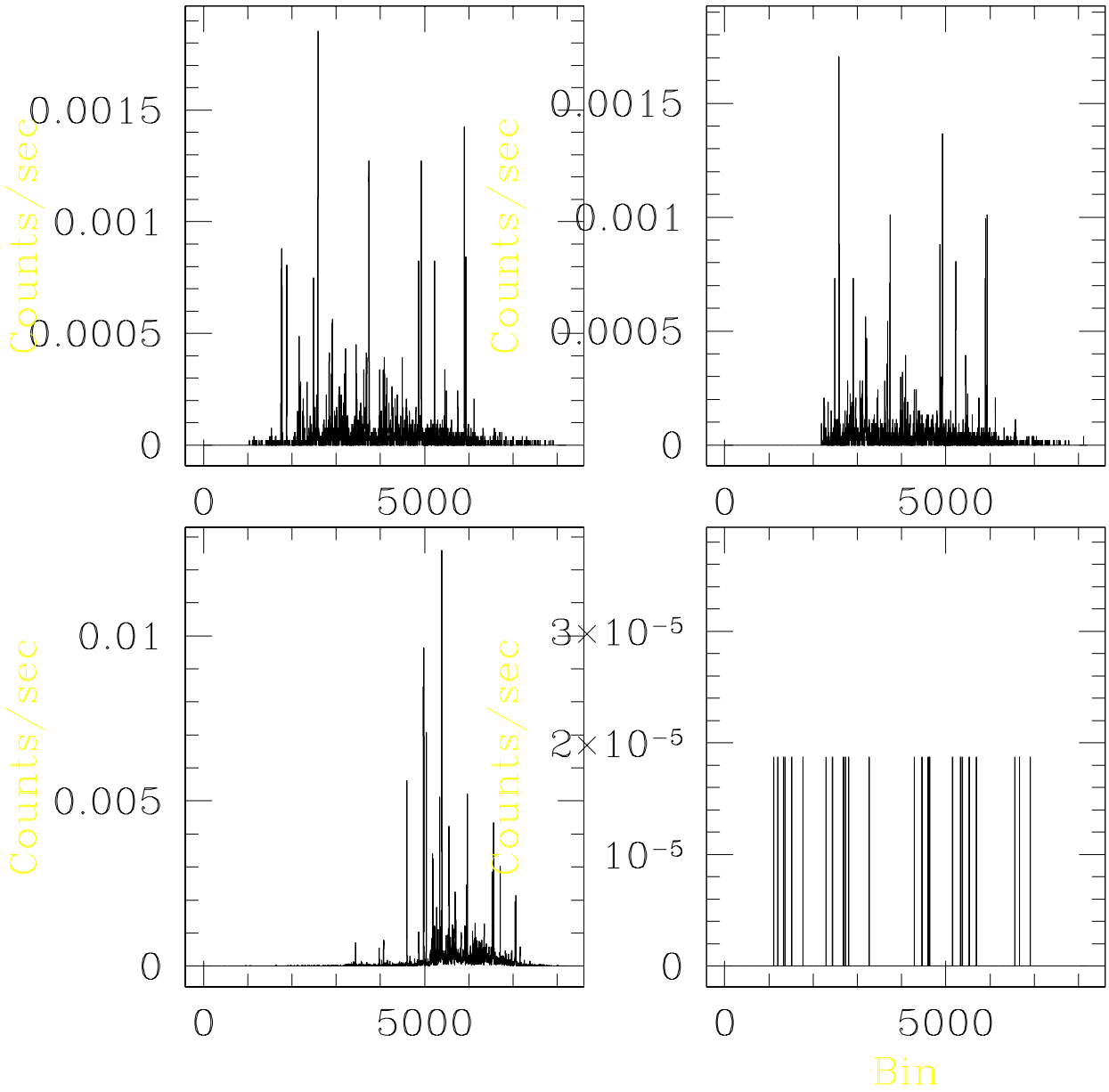
Example - calling Sherpa & ChIPS

/data/ciao_demo/threads/S-Lang/hetg_plot/

S-Lang code can use both Sherpa and ChIPS commands, if called from a Sherpa session. Here we define a function that plots the +/- 1 orders of the HEG and MEG (if loaded).

```
define hetg_plot () {

    if ( _NARGS != 0 ) {
        message("Usage: hetg_plot()\n" +
            "makes pretty plots of grating data");
        return;
    }

    () = sherpa_eval("set plot noerrorbars");

    % plot +/- 1 orders of HEG and MEG
    % - check the return code
    % to see if everything is a-okay
    variable err = sherpa_eval(
        "lp 4 data 3 data 4 data 9 data 10 data");
    if ( err != 0 ) {
        message("Error plotting grating data " +
        "... is it loaded?");
        return;
    }

    % make the plot look a bit better
    % - ignoring return codes here (lazy)
    () = chips_eval("split maj Y");
    () = chips_eval("split 2 2");
    () = chips_eval("split gap X 0.1");
    () = chips_eval("split gap Y 0.05");
    () = chips_eval("redraw"); % replot
    return;
}
```

```
unix% sherpa --slscript hetg_plot.sl
...
sherpa> data acis_pha2.fits
...
sherpa> hetg_plot
```



Example - handling arrays

/data/ciao_demo/threads/S-Lang/loops/

Since S-Lang resembles C in many ways, numerical algorithms can often be converted with little effort. The following code, which calculates the sum of all the elements in an array, is essentially unchanged:

```
% calc_sum1() computes the sum of elements of a
% 1-D array.
%
define calc_sum1 (x)
{
    variable num = length(x);
    variable total, i;

    total = x[0];
    % start looping at the second element
    for ( i = 1; i < num; i++ )
    {
        total += x[i];
    }

    return total;
}
```

In use it looks like

```
unix --slscript calc_sum1.sl
chips> x = [1:10]
chips> print(calc_sum1(x))
55
chips> y = [ [1:5], [6:10] ]
chips> print(calc_sum1(y))
Invalid Parameter: Array requires 2 indices
Invalid Parameter: print(calc_sum1(y));
```

However, the real power of S-Lang is revealed if you re-write the algorithm:

```
% this is faster than calc_sum1, and will handle
% numerical arrays of any dimensionality
%
define calc_sum2 (x)
{
    variable total = 0;
    foreach ( x )
    {
        total += ();
    }

    return total;
}
```

The foreach() command loops through every element in the supplied array - whatever its dimensionality - placing the current value onto the stack. This value is then retrieved - and added onto the current total - by the 'total += ();' line.

```
chips> x = [1:10]
chips> y = [ [1:5], [6:10] ]
chips> () = evalfile("calc_sum2.sl")
chips> print(calc_sum2(x))
55
chips> print(calc_sum2(y))
55
```

The Varmm library provides min(), max(), sum() and reverse().

CIAO

Example - array manipulation

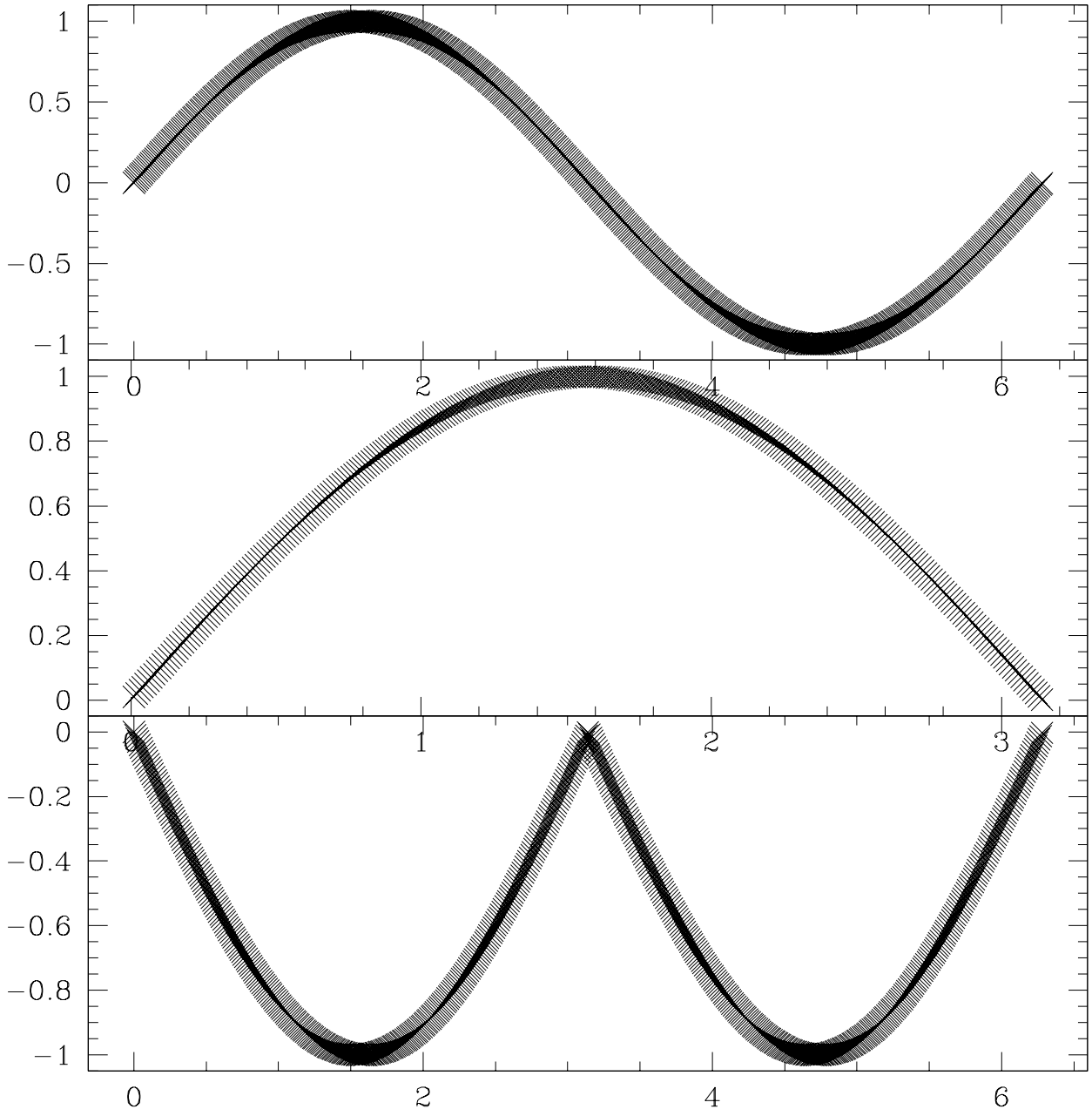
S-Lang contains a number of mathematical functions (eg cos, asinh) that work on arrays as well as scalars. It is also possible to access/change subsets of an array using the array indexing functions:

```
chips> x = [0:2*PI:0.01]
chips> y = sin(x)
chips> split 3
chips> () = curve(x,y)
chips> i = where(y > 0)
chips> d 2
chips> () = curve(x[i],y[i])
chips> y[i] *= -1
chips> d 3
chips> () = curve(x,y)
```

www.s-lang.org/doc/html/slangfun-6.html
www.s-lang.org/doc/html/slang-11.html

Also see the following scripts on the "Download Scripts" section of the CIAO web pages:

analyze_ltcrv.sl
  lc_clean.sl
  regions.sl
  sstats.sl


1
0.5
0
−0.5
−1
0
2
4
6
1
0.8
0.6
0.4
0.2
0
0
1
2
3
0
−0.2
−0.4
−0.6
−0.8
−1
0
2
4
6

There are many more subjects that could be mentioned, for instance:

- ChIPS, Sherpa, and the Varmm have resource files which can be used to customize your S-Lang environment.
- The behavior of ChIPS and Varmm can be controlled using "state objects".
- Varmm uses structures to store/write data files. Structures can be created "on the fly" and are useful for grouping together data.
- Associative arrays: these are similar to normal arrays, except that strings, not integers, are used to index the elements.
- I/O, string manipulation, running/accessing system commands, ...

Useful resources:

http://www.s-lang.org/
ahelp slang-tips (examples are in /data/ciao_demo/threads/S-Lang/tips)

S-Lang in CIAO by Example (coming soon)