Splat!

a data analysis program

Version 5.3
January 2012

User's Guide

Contents

Chapter 1

Program Description

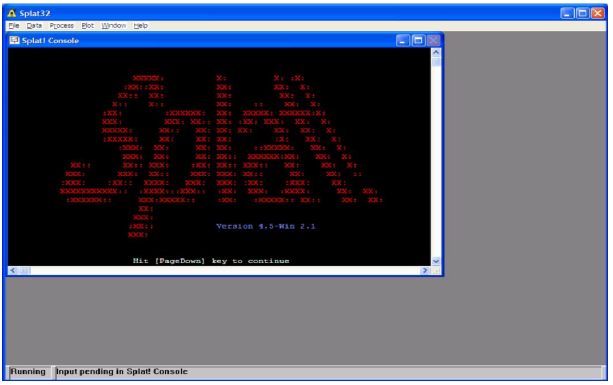
Splat! is a general purpose data analysis program. It uses a simple, intuitive command line interface to input, manipulate, and display data. Think of it as a scientific calculator for processing computer files- it can preform many of the same data manipulation functions, and can produce primitive graphical output, but like a calculator the output options are a bit limited.

Splat! is not a spreadsheet program like Excel (^{TM}Microsoft Corp.). It is also not a graphical plotting program like Origin (^{TM}OriginLab) or IGOR (^{TM}Wavemetrics). Nor is it a symbolic mathematical program like Mathematica (^{TM}Wolfram), Maple (^{TM}Maplesoft), and Mathcad (^{TM}Mathsoft). To illustrate what Splat! is, rather that spend a lifetime describing what it is not, it is probably best to run though a quick example.

1.1 Example Splat! Session

Let us assume we are interested in finding out how the cost of textbooks scale with how big the book is. Our data set is a set of X, Y values for each book: the number of pages (X) and the cost (Y). Our data is stored in two text files (standard human readable text files) called `soft.txt` (for soft cover books) and `hard.txt` (for hardcovers). We start off or analysis by firing up Splat![1]. Here is what the Windows XP version looks like:

[1]See Chapter 4 on the details of how to install Splat! for your particular operating system.



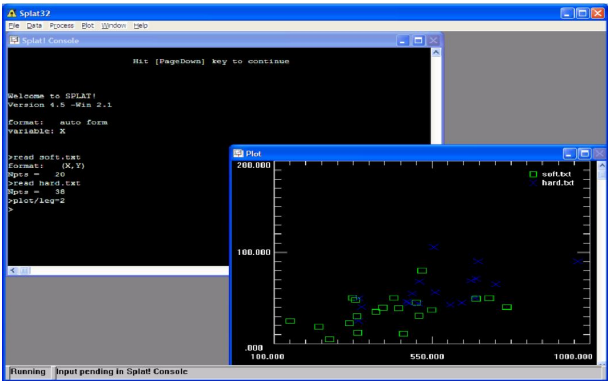
After pressing the <Pg Dn> key, we have Splat! read in the two files using the **READ**[2] command: at the > prompt we enter: `READ soft.txt`, and `READ hard.txt`. The first file contains 20 data points, the second file contains an additional 18 data points (for a total of 38 points). Now that we have read the data into Splat!, we can take a quick look at the data using the **PLOT** command. This is as simple as entering
cliPLOT at the >prompt, or by selecting `Plot` from the Plot pull-down menu (in the Windows version). However, to make things a bit more complicated we also want to display a plot legend in the upper right corner, this is accomplished with `PLOT/LEG=2`.

In the DOS version the text command line display is replaced by a full screen graphics display until the user presses <enter>. In the Windows version the plot shows up in a new window behind the command console window. Pressing <tab> switches the focus between the two sub-windows[3].

Note that the data points from the two different input files are displayed with different symbols. This looks nice, and makes the output easier to understand, but as far as Splat! is concerned all data points are treated equally regardless of their source. Only in

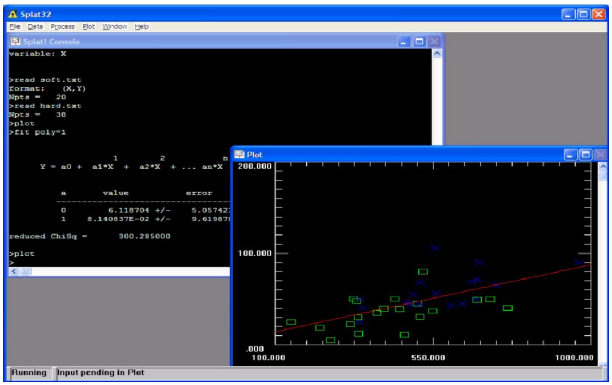
[2]In this manual Splat! commands are indicated in bold all caps when mentioned in the text. This is simply so they stand out, the Splat! command line interpreter is case insensitive. Sample input/output lines are displayed in a fixed width font (usually in all caps).

[3]You can also use the mouse to select the plot window; but, this is not recommended. If you do use the mouse, you must also use the mouse to re-select the Splat! console sub-window (<Tab> will not work).



plotting is there any distinction made (and only as long as the order of the data points is maintained).

Looking at the data, there is only a slight dependence on text book price with page count. We can further quantify this by fitting a line to the data and extracting the slope (with units of dollars per page). To do this we **FIT** a first order polynomial: `FIT POLY=1`. The $a1$ coefficient is the slope we are after. In addition to the fitted coefficients displayed in the table, Splat! also produced a line output for the plot. If we replot the data by typing `PLOT` we can see how well this line fits the data.



In addition to polynomials, Splat! has a number of other func-

tions that can be used in fitting, but they don't seem warranted in this case. Splat! was designed to handle experimental data with error bars, but our text book data doesn't have any, so the uncertainties reported in the fitted coefficients and the goodness of fit parameter (reduced χ^2) have little meaning.

In addition to fitting, we can also get statistics on a single variable. For example, if we are interested in the cost of books, we can get a number of statistical number with the **STAT** command: `STAT Y` (recall the y-column is the cost).

```
>stat y


Statistics for   38 data points:

Average   =       46.655790
Avg. Dev. =       15.935350
Std. Dev. =       22.039090
Variance  =      485.721300
Skew      =    5.434262E-01
Kurtosis  =    2.412953E-01
               (mesokurtic )

Minimum =         4.950000
Maximum =       105.750000
Median  =        44.950000
Sum     =      1772.920000
```

Since most of the sampled books were purchased long ago, the average price is quite low compared to present textbook prices. Good thing Splat! is free.

When we are all done with our analysis, we **EXIT** the program with the `QUIT` command[4] Other than outputting a new data file, there is no way to save your work in Splat!. Hence, when you quit there is no prompt to save your work before exiting.

1.2 Program Uses

Splat! is generally used to either manipulate 'raw' data into a format for another program (such as a fancy plotting program), or to

[4]A sad attempt at verbal judo. Both `EXIT` and `QUIT` close Splat!. Many other Splat! commands also have multiple aliases.

extract some useful information from a data set.

Data Manipulation One of the (few) strong points of Splat! is its averaging capabilities. While averaging, Splat! can create statistical error bars or do weighted averaging of data with pre-existing error bars. If so directed, Splat! can also recognize and discard outliers/bad data points while averaging. Complex mathematical transformations can also be applied to the data. Simple commands can also be used to rescale data.

Data Analysis As illustrated in the previous section, Splat! has a number of commands used to extract information from xy and single variable data sets via fitting and statistical analysis.

To support these primary functions, Splat! has a number of tools for importing and exporting data, and displaying data. Additional commands control how the program displays and interprets data.

1.3 Program Organization

Splat! has three main components that you the user should be aware of:

- the command line interpreter
- the data storage arrays
- internal configuration flags

1.3.1 Command Line Interpreter

You interact with Splat! via a command line interface. The Windows version of Splat! is essentially a shell using menus and dialog boxes to create the equivalent text-based commands. Commands are of the form:

```
verb[/optional qualifiers] parameter
```

Most basic operations in Splat! can be accomplished without any qualifiers, and even the parameters are often unnecessary for 'default' conditions. While these default conditions are appropriate in most cases, they may not be the best for your particular application in which case you a have a bit more typing to do.

Note of caution:
Splat! assumes you (the user) is reasonably intelligent and knows what you are doing. Hence, there is little error checking built into the program. It is very easy to have Splat! attempt to analyze non-existent data, or have it attempt to divide by zero (which occurs in many incarnations). Naturally this will crash the program. It is recommend that you save your work before attempting complicated operations such as fitting or math operations.

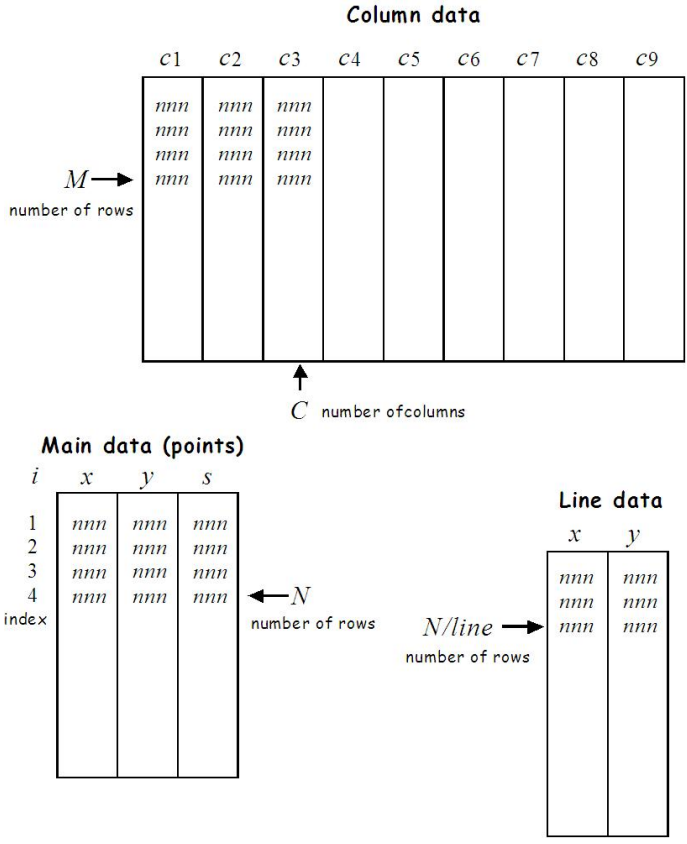
Splat!'s command line interpreter is case insensitive, and most commands can be abbreviated significantly. For example, the **AVERAGE** command will be executed for any command that begins with an `AV`, thus all of the following are acceptable: `AVERAGE`, `AVE`, `AVG`, `AV`, `av`, `aVerySilly`, ...

1.3.2 Data Storage Arrays & Internal Flags

Splat! has three main data storage arrays: (*i*) the standard *xys* array, (*ii*) the column data array, and (*iii*) the *xy* line data array.

Most commands either work exclusively on the *xys* data or use these as the default. The *xys* data is plotted as points. A separate *xy* array holds data that will be plotted as a line. While it is possible to transfer data between these two arrays, this is somewhat awkward. It is much easier to transfer data between the standard *xys* data array with the column data worksheet.

Each of the three array has a maximum size of about 50,000 points (you can find out this size with **HELP**). Each array has a flag that tells how many rows are currently being used. For the *xys* array this is called `N`, for the column array it is called `M`. A further variable `C` keeps track of how many columns are filled. The number of values in the *xy* line data is also called `N`, but this `N` is different than the `N` of the point data. Confused? Dont worry, Splat! generally keeps track of all these variables for you. When



you reset the program using the **CLEAR** command the program doesn't erase your old data, it just sets the number of points `N` to zero. Using the **MAKE** and **SET** commands you can directly set the values of these counters.

The **FORM** flag informs Splat! about the type of data in the xys array. In `form=1`, only a single column of data contains useful data. In `form=2` either the x (or y) column contains data and the s column contains the uncertainty/error bar for the data column. More typically one uses either `form=3` which is for xy data or `form=4` for xy data with the uncertainty in y stored in column s. In `fom=7` the third column is relabeled as z, allowing plotting of xyz data in 3D.

How does this all work? Recall the example in section 1.1 of a

data set consisting of the data pairs of page counts and book costs. When you **READ** in a file with two columns of data Splat! defaults to assuming the columns correspond to the values for x and y. If you use the **LIST** command to look at the data Splat displays only the x and y columns from point 1 to `N`. To be more concrete, let us consider an example where our data consists of the following 9 values:

```
1 10
2 15
3 8
1 12
2 13.5
3 16
1 11
2 11
3 17
```

If we **READ** this into Splat! and **LIST** the data, Splat! displays only the x and y columns of data from 1 to 9:

```
Welcome to SPLAT!
Version 4.5 -Win 2.1

format:   auto form
variable: X


>read sample.dat
format:   (X,Y)
Npts =    9
>list

          1        1.000000        10.000000
          2        2.000000        15.000000
          3        3.000000         8.000000
          4        1.000000        12.000000
          5        2.000000        13.500000
          6        3.000000        16.000000
          7        1.000000        11.000000
          8        2.000000        11.000000
          9        3.000000        17.000000
>
```

If we now proceed to **AVERAGE** the data, the three values at each x value will be averaged together. If we list the data now,

we only have three data points. When Splat! averaged the data, however, it calculated the error in mean for each value and stored the value in the s column. Splat! doesn't know if we want to use this information, so it is 'hidden' when we **LIST** the data.

```
>
>avg
Npts =     3
>list

          1        1.000000        11.000000
          2        2.000000        13.166670
          3        3.000000        13.666670
>
```

But if we switch to `FORM 4`, the newly created error bars are also listed,

```
>
>form 4

format:   (X,Y,S)
variable: X
>list

          1        1.000000        11.000000    5.773503E-01
          2        2.000000        13.166670        1.166667
          3        3.000000        13.666670        2.848001
>
```

While not shown, if we now plotted the data each data point would include a y error bar.

Chapter 2

Command Overview

As mentioned in section 1.3.1, Splat! commands use the following syntax:

```
Verb/qualifier(s)   parameters
```

All qualifiers and most parameters are optional. If you omit a required parameter, Splat! will prompt you for it. Unless the parameter required is obvious, you can usually type in a `?` to get a list of available options. In this documentation, optional parameters and qualifiers are enclosed within brackets.

One qualifier that is supported by many commands is the range option, `/n1:n2`. This limits the scope of the Splat! command to only those points with an index value between n1 and n2. The 'index value' is not the value of x or y but the counting index that runs from 1 to `N`. For example, `LIST/4:8`, would only list data points 4 through 8. If the first number is omitted, the lower limit defaults to 1, while if the second number is omitted, the upper limit defaults to the number of data points. Except when used with **PLOT**, the numbers can be replaced with an expression including the variables `N` and `M`. For example, if you only want to look at the last 11 data points in a list of 100, you could use any of the following:

```
LIST/90:100
LIST/90:
LIST/n-10:n
LIST/n-3!-2*pi+e:n
```

There are a lot of Splat! commands. A few are far more useful than others. Here is the big picture:

Loading/Saving data:	**READ**, **WRITE**, **ENTER**
Examining data:	**LIST**, **BIN**, **PLOT**
Plot control:	**PLOT**, **POINT**, **LINE**, **MARK**, **LABEL**, **LEGEND**
Analyzing data:	**STAT**, **FIT**, **D**, **DD**, **INTEGRATE**, **REGRESS**
Processing data:	**MATH**, **FFT**, **SORT**, **AVERAGE**, **NORM**, **SMOOTH**
Moving data:	**ADD**, **PORT**, **EDIT**, **MATH**, **SWAP**
Controlling Splat!:	**FORM**, **SET**, **MAKE**, **CLEAR**, **CLS**
Worthless features:	**CALC**, **YO**, **ECHO**
Worthwhile features:	**HELP**, **CD**, **DIR**, **$**

Detailed information on each command can be found in the Chapter 3. Here we present a brief overview of useful commands in each main area.

2.1 Entering Data and Moving it Around

There are three main routes of getting some data into Splat!, (*i*) reading it in from a file, (*ii*) entering it from the keyboard, or (*iii*) creating it internally from a function. Let us look at all three routes a bit more closely.

2.1.1 File input

Splat! can read in most standard ASCII text files. Numbers on a line of data can be separated by spaces, tabs, or commas. Typically, Splat! starts up in `form=5`, auto format. In `form 5` Splat! starts scanning your input file for a line starting with numbers (hence it skips over most lines starting with characters) and figures out how many separate numbers are on each line. So for example, if the first line of the file is

```
12.00   452   400   7.5   1.0   20
```

Splat! sets `C` (the number of columns) to 6. And reads the data into `c1` through `c6` of the column worksheet. It then transfer the first three columns to the xys data sheet ($c1 \rightarrow x$, $c2 \rightarrow y$, $c3 \rightarrow s$), and sets `N`, the number of xys data points, equal to `M`, the number of column data points (which would depend upon how many of lines of data are in the file). Finally, Splat! changes the format from `form=5` (auto format) to `form=4` (xys). Note that Splat! only reads data into the column worksheet when it is in `form 5` (auto format) or `form 6` (column format). Since Splat! switched from `form 5` to `form 4` after the first read operation in the above example, any additional read commands will only attempt to read in three columns of data into the xys data array.

If the first line of data instead consisted of only two numbers, ie.,

```
12.00   452
```

Splat! sets `C` to 2. And reads the data into $c1$ and $c2$ of the column worksheet. It then transfer the first two columns to the xys data sheet ($c1 \rightarrow x$, $c2 \rightarrow y$), and sets `N`, the number of xys data points, equal to `M`, the number of column data points. Finally, Splat! changes the format from `form=5` (auto format) to `form=3` (xy).

If this all seems confusing, don't panic. Splat! is simply trying to make life easy for you. Further information can be found under the **READ** and **FORM** commands in chapter 3. Since Splat!'s **READ** command supports wildcards and multiple input files in a single operation, it is possible to avoid a lot of problems (especially with multi-column data files) by reading in all the input files in a single step.

Alternatively, if you know how many columns of data you have, you can specify the format before reading in the data. For multi-column data input you specify `FORM 6` (multi-column) and fix the number of columns with the **SET** command (i.e. `SET C=6`).

2.1.2 Keyboard Entry

A second way of getting data into Splat! is to type the data in by hand using the **ENTER** command. Typically you specify the form beforehand (with **FORM**), otherwise Splat! defaults to assuming you want to enter xy data (`FORM 3`). This method of data entry

is pretty basic, and not recommended for entering more than a handful of points. You are far better off typing the numbers into a file using your system's text editor. In fact, Splat! uses the system's default text editor for the **EDIT** function which allows you to manually edit the input.

2.1.3 Creating Data from a Function

Another option for generating data is to specify the values using a function. As an example, let us assume we want 500 points of data ranging from x=0 to x=10 with y values of $\sin(x)$. To do this in Splat! we first create 500 points of data (from thin air) with the **MAKE** or **SET** commands, i.e. `MAKE 500` or `SET N=500`. Next we need to set the x values. To do this we use the **MATH** function with the index i (which runs from 1 to 500) using `MATH X=10*(I-1)/(N-1)`. With the x values set, we next turn our attention to the y values, `MATH Y=SIN(X)`. Finally, we switch to `Form 3` so that Splat! knows to display only the xy values, `FORM 3`. With the Windows version of Splat! this whole process is automated for you by using the 'Create from function...' selection under the Data pull down menu. Column or line data can be generated in a similar fashion.

2.1.4 Moving Data Around

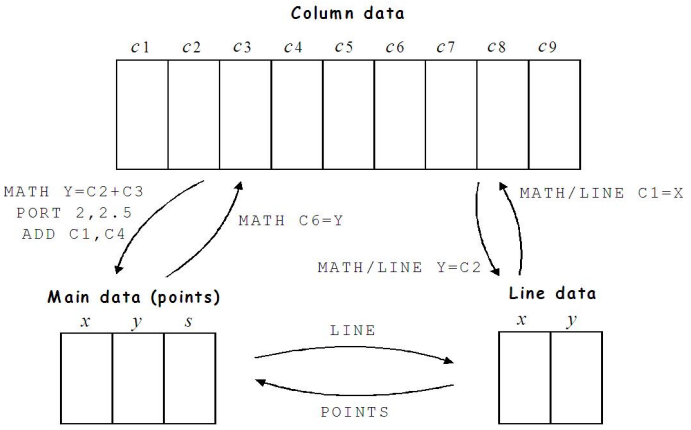
Suppose we had a data file with the following format

x-value $y1$-value $y2$-value $y3$-value.

We want this data to be interpreted as three xy pairs, i.e. $(x, y1)$, $(x, y2)$ and $(x, y3)$. Unfortunately, Splat! doesn't know this and has instead (in auto format mode) read the x-value into $c1$ (and x), the $y1$-value into $c2$ (and y), the $y2$-value into $c3$ (and s), and the $y3$-value into $c4$. It also picked `form 4` (xys) for the data format. Some of this is right, but alot of it is not what we want- what a mess. Relax, Splat! has some simple commands to move data between the three data arrays. To fix up this particular example case we do the following,

`FORM 3`	Switch to xy format
`SET N=0`	Erase the existing messed up xys data
`ADD C1,C2`	transfer the $x, y1$ pair
`ADD C1,C3`	transfer the $x, y2$ pair
`ADD C1,C4`	transfer the $x, y3$ pair

In addition to the **ADD** command, both the **MATH** function and the **PORT** command can be used to transfer data from the column array to the xys data set. Only the **MATH** command can be used to transfer data back to the column array from the xys data set. The **POINT** and **LINE** functions are likewise used to shuffle data between the xys data set and the xy line data. See the appropriate entries in the Command Reference (chapter 3) for detailed information. The **EDIT** command, with the correct qualifier, can be used to edit any of the three data sets.



2.2 Analyzing Data

Once you have read some data into Splat! you generally want to do one of three things with it, (*i*) look at it (i.e. plot it), (*ii*) modify it in some way (i.e. average it) or (*iii*) extract some information from it without modifying it. The following two sub-sections deal with the first two options, this sub-section deals with the latter.

Splat! can analyze single variable data with either the **STAT** command or the **HIST**ogram command. Both are pretty lame. You are far more likely to use the **FIT** command to analyze xy or xys data sets. The **FIT** command can be used to perform either a linear least squares fit or a non-linear least squares fit to the data, the choice depends on the function being fit. Fitting a polynomial is a linear fit (i.e. the fitted coefficients are linear combinations of the fitted basis), whereas fitting a Gaussian peak is a non-linear fit. With the important exceptions of problems with round-off errors linear fits and pathological data linear fits will always yield the 'best fit' answer. Non-linear fitting, on the other hand, requires an initial guess of the fitted coefficients before narrowing in on the 'best-fit' values. If you start with initial guesses that are far from the 'best-fit' values, the fit may not find the global 'best-fit' and instead only a local one. Splat! attempts to come up with reasonable starting values based on input data. For example, when fitting a Gaussian peak Splat! assumes the largest y-value corresponds to the peak amplitude, the x-value of this point is the center of the peak, and the width is the x-value where the amplitude falls to half the maximum amplitude. This should work most of the time, but not always. Since Splat! automatically produces a line of the fitted function, it is very easy to compare the fitted function with your data using the **PLOT** command. This visual inspection procedure is *highly recommended* to verify that Splat! has done a reasonable job before accepting non-linear fit results.

When Splat!'s fitting fails

In cases where Splat! has done a poor fitting job, you are in a bit of a pickle since Splat!'s fitting procedure is fairly automated. Nonetheless, you still have a few options:

1. Change that starting conditions by editing the data to remove bad points, or otherwise limiting the data range.
2. Switch fitting functions. If fitting a Gaussian peak doesn't work, maybe a Gaussian peak with a constant background (or vice versa) would work better. Likewise you could switch between a polynomial and fixed power of x.

3. Simplify the problem for Splat!. Sometimes Splat! fails when the input values are way too Big or too small; in these cases you may be able to get good results by simply rescaling the x or y values by a fixed amount and refitting. As another example, imagine you are interested in finding the area of Gaussian peak, but `fit Gaussian` is producing a poor fit. If you know the expected width of the Gaussian peak, you can simplify Splat!'s fitting job by reducing the number of free parameters by supplying the known width (e.g. `fit Gauss=0.25`). Another trick for fitting Gaussian peaks when the background is neither zero (`Fit Gauss`) nor a constant value (`Fit Gaussb`), is to first fit a straight line to the data (ignoring the Gaussian peak) to remove the background (i.e. `FIT/SUB POLY=1`), then fitting a Gaussian (with background) to extract information on the peak (i.e. `FIT GAUSSB`).

Some basic calculus operations can also be performed on the data without modifying it (too much) including **INT**egration and derivatives (**D** for the first derivative, **DD** for the second derivative). These are both pretty lame: the integration routine also sort the data, and only uses the simple trapezoid rule. The derivative functions are highly sensitive to noise (and round off error).

2.3 Processing Data

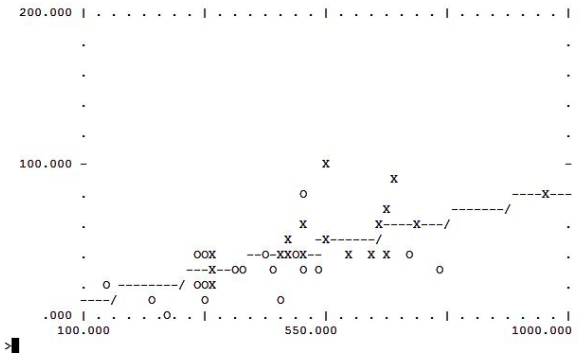
Splat! includes many commands to modify the *xys* data in some fashion. Some of these commands can also be applied to the column data of *xy* line data with the appropriate qualifier. By far the most useful of these commands is the **MATH** function. This function can be used with wide variety of standard mathematical functions and operations to express any one variable (`X,Y,S,C1,C2,...C9`) as a function of itself and any combination of the others as well as the index `I`, the total number of points (`N` or `M`), and the mathematical constants `E` (e=2.718...) and `PI` (π=3.14159...). See the **MATH** entry in the Command reference for numerous examples.

A related function is the **NORM** command which can be used to rescale data in one easy step (including error bars if in *xys* `form`

4). Both **MATH** and **NORM** modify values with out changing either the total number of points or the order of points in the data arrays. In contrast, **SORT** can be used to reorder the data. **AVERAGE** both sorts the data and reduces the number of data points. **SMOOTH** is the less intellectually honest, bastard offspring of **AVERAGE** that does some averaging of noisy data without decreasing the number of data points. The fourth circle of Hell is reserved for those who use this function. Also destined for trouble are those who venture to use the Fast Fourier Transform (**FFT**) function that only sort-of works.

2.4 Plotting Data

Ah, back in olden times[1] the primary output device for Splat! was a text only VT-52 terminal. Splat!'s **PLOT**ting functions have progressed quite a bit since then, but the old-school text output plotting routine can still be accessed with `PLOT/T`. Here for example is the text version of the plot from page-6:



The modern version of **PLOT** has oodles of optional qualifiers. These fall into three broad classes, (*i*) system-dependent, graphics-mode selection controls, (*ii*) options that control plot symbols and line types, and (*iii*) other qualifiers that affect how the plot looks. Category (*i*) qualifiers are best left to the to the detailed command reference (chapter 3) and the program installation/setup chapter

[1] Also known as the early 1990's.

(Chap. 4). Category (*ii*) qualifiers are very user unfriendly and best not accessed directly. Instead, it easier to adjust these parameters though the **MARK** and **LABEL** commands. Or, even easier, let Splat! worry about them with the `I_MARKER=1` option selected in the `SPLAT.INI` file (see Sec. 4.5). Indeed, considering the only one to see the output of Splat! is the user, there little reason to mess around with how points and lines are displayed and labeled. Far more useful are parameters in category (*iii*) such as `/XLOG` and `/YLOG` that select a log scale instead of the usual linear scales. The `/DATA` qualifier prevents Splat! from rounding the plot limits to nice 'round' numbers and instead forces the limits to the actual data limits. This same qualifier can also be used with the **FIT** command to limit the fitted line (useful when 'round' numbers are unphysical (i.e. would be zero on a log scale).

Chapter 3

Command Reference

$command
In the DOS version, this spawns a DOS process to execute the given command. In the Linux version this spawns a process to execute the given unix command. In the Windows version, this command is no longer used since you can switch between windows easy enough.

ADD`[/n1:n2] Cn1 [,Cn2, Cn3]`
Transfers data from column data set to the points `(X,Y,S)` data set. The exact destination is set by the format and default options. For example, in `form=3`, `ADD C1,C5` will append the `C1` data to the `X` data, and append the `C5` data to the `Y` data. See also the **PORT** command.

AVERAGE`[/SUM,/SD,/D,/X] [delta_x]`
Averages all `Y` values with the same `X` values (or within `delta_x` wide bins). In form=3 (xy mode) the `S` vector is set equal to the uncertainty in the mean ($\sigma/\sqrt{N}$). If you instead want the 'real standard deviation, use the `/SD` option. Note that you must use **FORM** to see these 'new' `S` values. In form=4 (xys mode), `S` is the weighted average. When using the `delta_x` parameter, the `X` coordinate is set equal to the simple average of the data points. In `/SUM` mode, all `Y` values within a bin are added together instead of averaging.

If the `/D` option is included Splat! will attempt to discard 'bad'

data points. . To see how this is done, consider the following example, suppose you had four points of 5, 5.1, 4.9 and 100. Clearly the 100 point is wrong, but this one bad point will shift the average from 5 to 29. With the `/D` option Splat! corrects for this by finding the median value and the standard deviation. Then Splat! discards all values more than 2.5 standard deviations from the median (a robust indicator of the mean). Or you can over-ride this default value by supplying a new cutoff (measured in units of the standard deviation). For example, `/D=3` would set the good/bad threshold at 3σ from the median.

In form=7 (xyz-mode) `Z` values at the same `X,Y` values (or within `X`$\pm$`delta_x`, `Y`$\pm$`delta_x` wide bins) are averaged together. With the `/X` option the binning criteria is only applied along the x-axis (i.e. `X`$\pm$`delta_x`, `Y`± 0 bins).

BIN`[/n1:n2] num-bins,[var]`
A simple text histogram of the default variable or the specified variable. Same as **HIST**.

CALC
Reverse Polish Notation (RPN) calculator mode. **CALC** has its own help command (or see **MATH**). In **CALC** mode, only the `N` variable is defined, `X,Y,S`, and `I` are not valid. Use `QUIT`, `EXIT` or <control>-Z to return to the Splat! prompt.

Expressions can be entered on one line in either algebraic mode or RPN mode. For example, $4+(2*3)$ can be evaluated by entering either `4+2*3`<enter> or `4 2 3 * +`<enter>. When using previous results (displayed on the stack) you must use RPN notation; so to add two number (i.e. 2 & 5) you would enter `2`<enter>, `5`<enter>, `+`<enter>.

CD `directory`
Changes the default directory (same as the MS-DOS/Linux command).

CLEAR
Sets the number of points to zero, no line, and default format and variable.

CLS
Clears the screen.

D`[/ADD, /n1:n2] [xwidth]`
Numerical derivative of the `Y` data (dy/dx). The derivative is found by fitting a line to all points within a sliding window of width `[xwidth]`, if this is not specified it defaults to a few percent of the maximum x range. If there are not enough points to fit a line within an interval, the size of the interval is increased. The x-spacing need not be constant, and can contain repeats. The output results are saved in the xy line array. This replaces any existing line unless the `/ADD` switch is included.

Note that numerical methods to calculate the derivative (**D**) and second derivative (**DD**) are highly sensitive to noise and any roundoff errors in the data. Results should be used with caution.

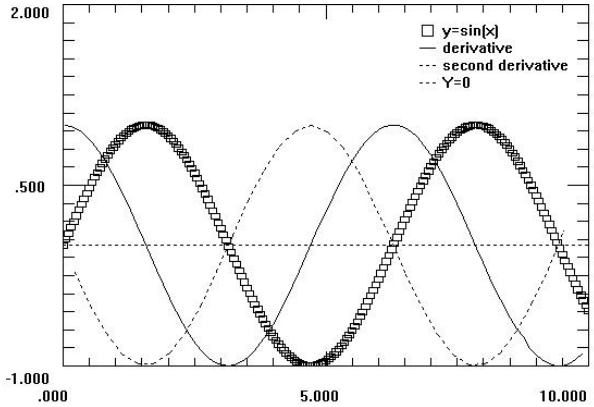


Figure 3.1: Sample derivative (**D**) and second derivative (**DD**) output of $y = \sin(x)$. Raw function has 200 data points, first derivative output has 81 data points, second derivative output has 32 data points. Note small glitch in first derivative curve near $x = 8.7$.

DD`[/ADD, /n1:n2] [xwidth]`
Numerical second derivative of the `Y` data (d^2y/dx^2). The derivative is found by first fitting a line to all points within a sliding window of width `[xwidth]`, if this is not specified it defaults to a few percent of the maximum x range (see first derivative func-

tion **D**). Next the derivative of this intermediate line is found, i.e. $\frac{d}{dx}\left(\frac{dy}{dx}\right)$. The final spacing between points is typically about twice `xwidth`. The results are saved as the line replacing any existing line unless the `/ADD` switch is included.

DIR
Disk directory.

ECHO `text_string`
Prints `text_string` on the screen.

EDIT`[/LINE` or `/COL]` Edit data using the standard system editor. For the DOS version this is the DOS editor, for the Windows version this is Notepad, for the Linux version this is the vi editor. Data is written to and then read from the file `SPDATA.TMP`. Use `/LINE` to edit the line data, or `/COL` to edit the column data. Note that the column data is written out with a maximum of four numbers on a line. So if you have 6 columns of data, the first four numbers will be on line-1, and the last two on line-2. The next record will start with four numbers on line-3,...

ENTER
Enter data into Splat! from the keyboard. Enter a / to exit.

EXIT
Exit program. You can also use **QUIT**, or hit <Control>-Z (End-Of-File).

FFT`[/INV] [var]` Takes the Fast Fourier Transform of a set of equally spaced data points. Splat! will round down the number of data points to the nearest power of two, and find the magnitude of the Fourier components. The transform is applied to: the default variable if in `form 1` (x), or `form 2` (xs); the `Y` variable if in `form 3` (xy) or `form 4` (xys); or these defaults can be overridden by supplying a variable name. If in xy or xys format, the x-axis is transformed to the proper frequency scale. The `/INV` option reverses the FFT out of the frequency domain. Unfortunately, this doesn't reproduce the initial function since phase information is lost in the forward FFT, as well as truncations in the number of data points.

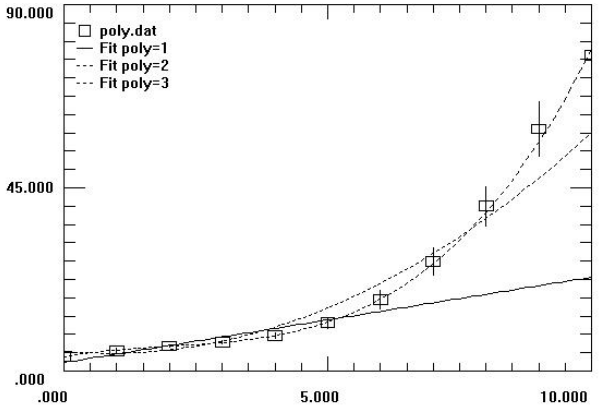
FIT[/options][/n1:n2] `function=n`
Options include:

`/D`	take line limits from data (instead of nice round numbers)
`/ADD`	append output line instead of replacing existing line
`/NL`	do not produce an output line
`/SUB`	subtract fit from data without producing an output line
`/n1:n2`	limit fit to selected range of data

Linear (or non-linear) least square fit a function to the data. Possible `functions` include:

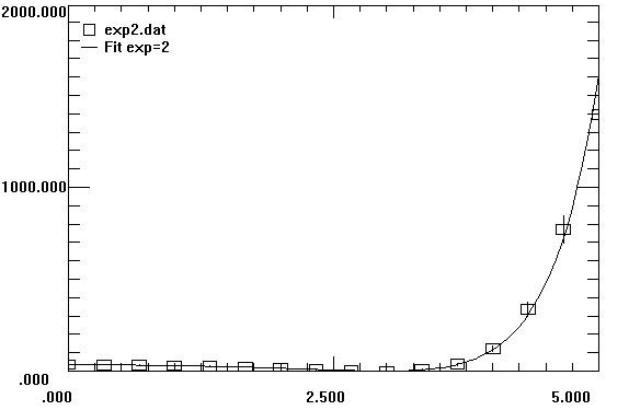
`FIT POLY=n`	n^{th} order polynomial	(linear fit)

$$y = \sum_{i=1}^{n} a_i \, x^i$$



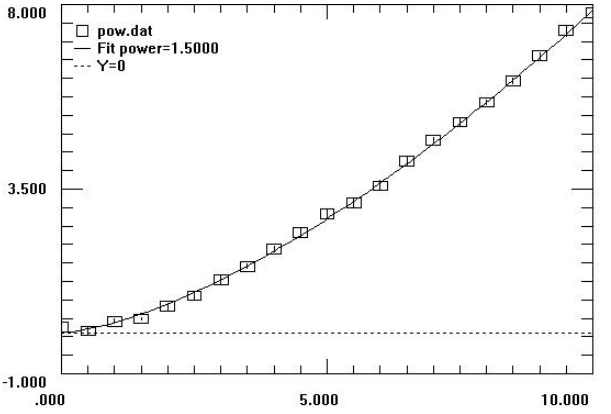
`FIT EXP=n` sum of exponential powers (linear fit)

$$y = \sum_{i=0}^{n} a_i \, \mathrm{e}^{i\,x}$$



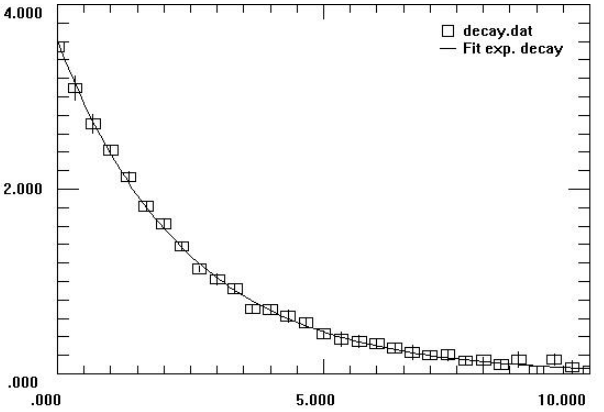
`FIT POW=p` n^{th} power of x (linear fit)

$$y = a_0 \; x^p$$



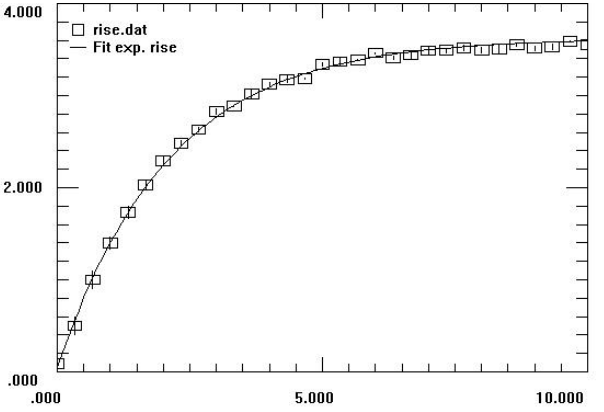
`FIT DECAY` single exponential decay (linear fit)

$$y = a_0 \, \mathrm{e}^{a_1 \, x}$$



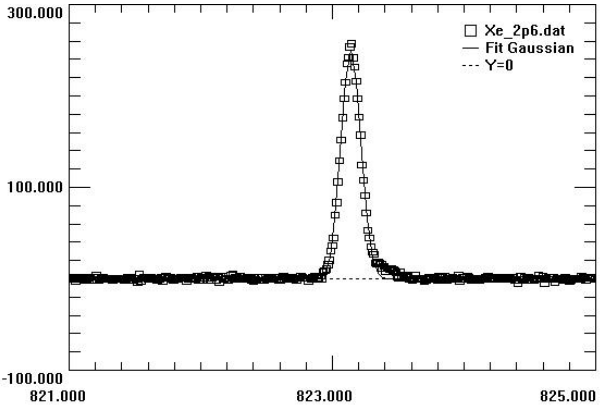
`FIT RISE` exponential rise (non-linear fit)

$$y = a_0 \left[1 - \mathrm{e}^{-\frac{(x-a_1)}{a_2}} \right]$$



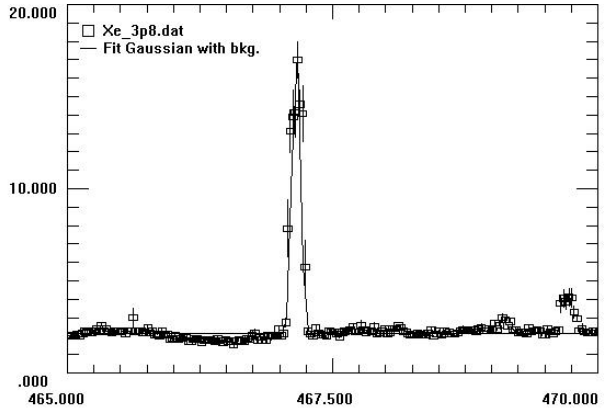
`FIT GAUSS[=fwhm]` Gaussian distrib. (non-linear fit)

$$y = a_0 \, \mathrm{e}^{-(x-a_1)^2 \, / \, a_2^2}$$



`FIT GAUSSb[=fwhm]` Gaussian dist. with constant background (non-linear fit)

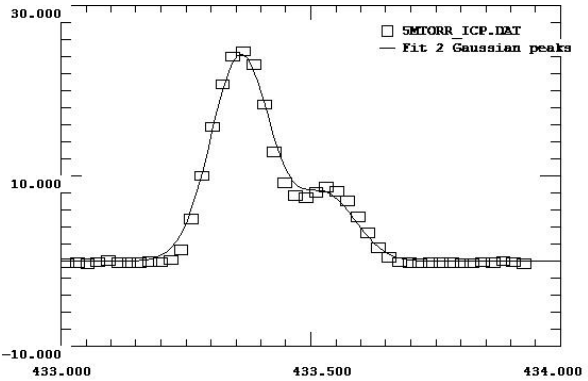
$$y = a_0 \, \mathrm{e}^{-(x-a_1)^2 \, / \, a_2^2} + a_3$$



`FIT GAUSS2[=fwhm] a1 a4`

Two Gaussian Peaks (non-linear fit)
at $x = a_1$ and a_4 with same FWHM

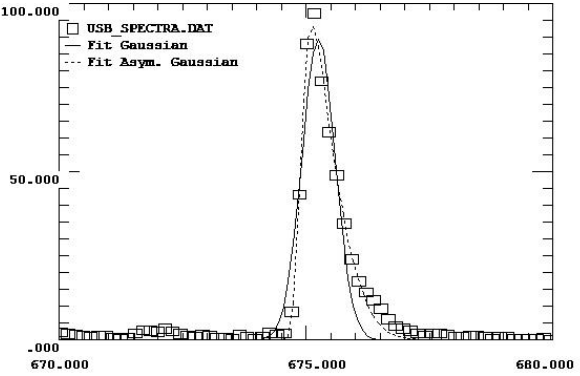
$$y = a_0\, \mathrm{e}^{-(x-a_1)^2\,/\,a_2^2} \;+\; a_3\, \mathrm{e}^{-(x-a_4)^2\,/\,a_2^2}$$



`FIT ASYMb[=fwhm]` Asymmetric Gaussian (non-linear fit)
dist. with constant bkg.

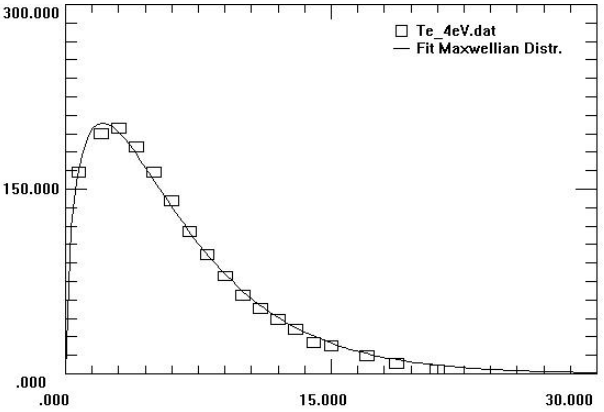
(Use `FIT ASYM` for fit without constant background.)

$$y = a_0\, \mathrm{e}^{-\left[\left(\frac{x-a_1}{a_2}\right)^2 \left(\frac{1+\exp[a_4(x-a_1)]}{2}\right)^2\right]} + a_3$$



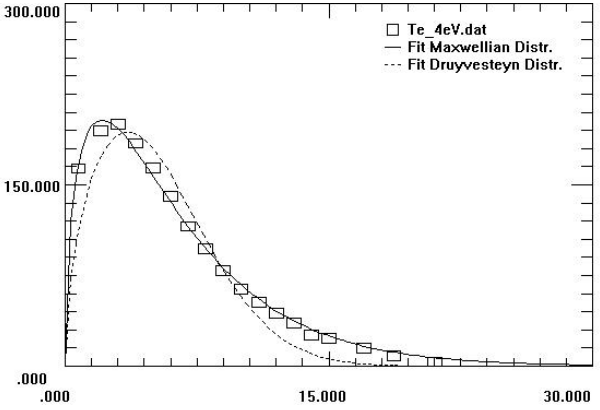
`FIT MAX` Maxwell-Boltzmann Distrib. (non-linear fit)

$$y = \frac{2}{\sqrt{\pi}}\, a_0\, \sqrt{x}\, \frac{1}{a_1^{3/2}}\, \mathrm{e}^{-x/a_1}$$



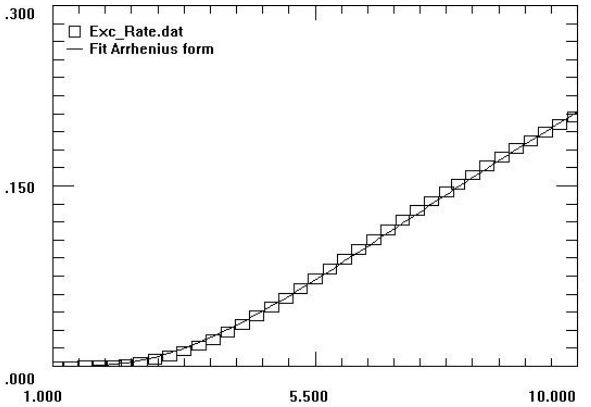
`FIT DRU` Druyvesteyn Distrib. (non-linear fit)

$$y = \frac{1}{\sqrt{\pi}}\, a_0\, \sqrt{x}\, \frac{1}{a_1^{3/2}}\, \mathrm{e}^{-0.243\, x^2/a_1^2}$$



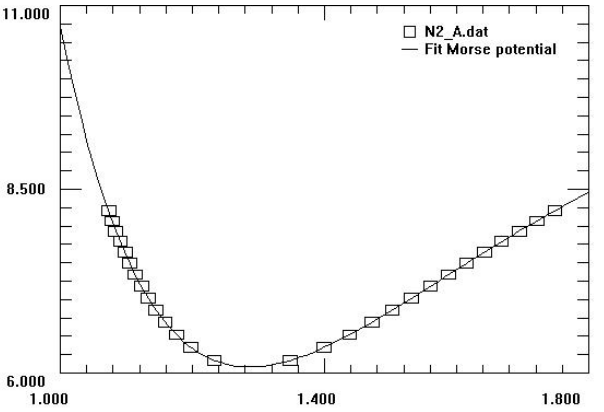
`FIT ARR` Arrhenius Form (non-linear fit)

$$y = a_0\, x^{a_1}\, \mathrm{e}^{\frac{-a_2}{x}}$$



`FIT MORSE` Morse Potential (non-linear fit)

$$y = a_0\, \left[1 - \mathrm{e}^{-a_1\,(x-a_2)}\right]^2 + a_3$$



Two forms of output are usually produced. (1) A text listing of the fitted parameters along with uncertainties; and (2) a 100 point line of the fitted function. Use the `/D` option to limit the output line to the range $x_{\min}$ to $x_{\max}$ of the data. To suppress the generation of the output line, use the `/NL` option. To append the fitted line output to your existing line data use the `/ADD` option.

Note that the uncertainties listed for the fitted parameters are only valid if the fit was done in form=4 with realisitc y-uncertainties. The text listing of parameters also includes the reduced χ^2 (χ^2 divided by the number of degrees of freedom) goodness of fit value. The smaller the value, the better the fit. A value of about one is expected for data with statistical errors bars when fit to its true functional form. In form=4 (xys-mode), the size of the y uncertainties are known, and the reduced χ^2 value has 'real' meaning. In form=3 (xy-mode), Splat! doesn't know the size of the y uncertainties, so these are set to 1 in the calculation of χ^2. The value of reduced χ^2 in this case can then be used to estimate the unknown uncertainties in y (i.e. switch to `form 4`, and use `math` to set the `S` values to give reduced $\chi^2 \approx 1$).

With the `/SUB` option specified, the fitted function is subtracted from the data and no line output is produced. By default, the fit is subtracted from all of the xy data, even if only a select range of the data (by using the `/n1:n2` option) was used in the fit. To limit the subtraction to only the fitted range, use the `/D` option.

The Gaussian fits also output the FWHM of the curve, and the area of the Gaussian portion of the peak. By default, the width of the Gaussian peak is allowed to vary as a free parameter. If, instead, you specify the width (i.e. `FIT GAUSS=1.1`) this is held constant.

FORM `format,[var]`
Change the default data format and variable. In general, the format number tells how many data columns are being used, while the variable tells Splat! what the default choice is for operations such as sorting. Valid options for format are:

format #	data format
0	user defined
1	X
2	X.S

format#	data format (continued)
3	X,Y
4	X,Y,S
5	auto format
6	multi column
7	X,Y,Z

Valid choices for the variable are `X`, `Y`, and `S` (or `Z` if in `form 7`). To change the variable with **FORM**, you must also provide the format. Examples:

`FORM=4`	change data format to form 4 (xys)
`FORM 3`	change to form 3 (xy)
`FORM 2,Y`	change to form 2 (xs) with `Y` as default variable (hence this is really (ys) mode...

Note that changing forms does not delete/reset the data int the `X`,`Y`,`S` vectors. It simply controls how many columns of data are displayed, and whether error bars should be plotted or used in some operations such as fitting or averaging, of if the third column of data should be interpreted as the z-coordinate (in form 7) for plotting, averaging.... The user defined format (0), auto format (5), and multi-column format (6) are special formats used to control how files are read into Splat!, see the **READ** command for more details.

HELP`[/MATH] [command]`
Help screen of Splat! commands. The `/math` option provides assistance on available math functions and variables. The help screen also lists the maximum number of data points your version of Splat! can support. The standard `HELP` command produces a list of most basic commands and options. More detailed information for some commands, particularly those with lots of options like **AVERAGE**, **FIT**, **PLOT**..., is also available with `HELP command`. For example, type `HELP FIT` for a list of options & parameters unique to the **FIT** command.

HIST
Histogram of data (See **BIN** for full details).

INCLUDE
Include/read data from file (see **READ** for full details).

INT `[x1,x2]`
Numerically integrates the y data, using the trapezoidal rule. The x-spacing need not be constant, and can contain repeats (which are averaged together). With no arguments, it integrates the entire data range. If both `x1` and `x2` are included, the integration is only performed over the points nearest the interval specified. Splat! will inform you of the actual range of integration. Note that this command also sorts the data.

$$\text{Answer} = \int_{x_1}^{x_2} Y_i(x)\, dx$$

LABEL`[/LINE][/N=num] text_label`
Used with **MARK** and **PLOT** to create a plot legend. With no options, the last marker set is labeled with `text_label`. Or use the `/N=num` option to replace the label for marker `num`. The `/LINE` option is used to select the line labels. Use `MARK/LIST` to view what labels are associated with each marker number. By default, Splat! sets labels to input filenames, *etc.*...

LEGEND `[0]`
In the Windows version only, opens a separate window with the plot legend. The window closes when you hit a key. `LEGEND 0` creates a static legend window that must be closed manually using the mouse.

LINE[/options] `[filename]`
Options include:

```
/NO
/CLEAR
/ZERO
/ADD
/n1:n2
```

With no parameters or qualifiers, this function converts the xy data to a line and sets the number of points to zero. If you include a `filename`, the line is read in from the file instead. Use the `/NO` or `/CLEAR` qualifiers to clear an existing line, `/ZERO` to add a $y = 0$ line. By default the present line is erased; to append the new line to the old line use `/ADD`. Line data is usually a two column affair (xy-coordinates with no error bars), but in `form 7` (xyz mode), lines have three columns.

LIST[/options]
Options include:

```
/LINE
/COLUMN
/FILE
/MARKERS
/n1:n2
```

With no options, this command lists the xys data on screen (with the number of columns determined by **FORM**). Use `/LINE` to see the line data, or `/COL` to see the column data. Use `/FILE` to see the filename used in the last read/write operation. The `/MARK` option lists the data markers and labels (see also **MARK**`/LIST`). Same as the **SHOW** command.

LOAD
Load/read data from file (see **READ**).

LS
Disk directory. Same as **DIR**.

MAKE `nsize`
Increases the number of data points by `nsize`. Use a negative number for `nsize` to eliminate points. See also the **SET** command.

MARK[/options] `[n1, n2, n3...]`
Options include:

```
/NO
/CLEAR
/LINE
```

/LIST
/ADD

Marks are simply points in the data list where you wish to change plot symbols. It is a simple way of automatically appending /NSET1=n1,n2,n3... onto each **PLOT** command. Marks are entered either automatically by Splat! (if I_MARKER=1 in the SPLAT.INI file), or by entering a list with the **MARK** command. Use the /NO or /CLEAR options to delete all marks, or use the /LIST option to view the currently set marks. If you want to add a new mark, without retyping in the whole list again, use the /ADD command. Any command that can alter the order of the data points (**AVERAGE**, **EDIT**, **SORT**, ...) will automatically clear all marks. Associated with each mark is a text label (visible when data is plotted with PLOT/LEG) that can be altered with the **LABEL** command.

A similar list is also kept for the line data (for appending /NLINE=n1,n2,n3... onto each **PLOT** command). Use the /LINE option to enter/alter/clear these values.

Outside of **PLOT** marks have no influence on how data is processed or displayed, all *xys* data are treated the same by Splat!.

MATH[/LINE][/n1:n2][/J] expression
Manipulate data. The expression must be of the form var=..., where var is either X,Y, S (or Z in form=7), or C1 to C9. With the /LINE option, you can alter the X,Y (and Z in form=7) line data (and C1 to C9). Note that you can alter a variable which is not in the current format (for example, if in form 3 (*xy*) you can still do MATH S= Y/10).[1] The part on the right side of the equal sign can be a rather complicated expression involving constants (numbers, π, e), variables (X,Y,S or Z,I,N,M, C1 to C9) and a variety of functions (sin, cos...).

variables/numbers:
X Y S Z I N M
-the X,Y, S (or Z) vectors, index I, and the Number of data points

E PI
- standard numerical values for e and π

[1] Z is *only* valid in form=7 (*xyz* mode), S is valid for all forms *except* form 7.

RAN RANG
- a random number between 0 and 1, a random number with a Gaussian distribution of width 1 centered at 0.

functions:
CHS
- convert to negative number

+ - * / ( ) INV ^ ** SQRT ABS
-algebraic functions

MOD INT NINT
-algebraic functions with integer values

MIN MAX
-Minimum and Maximum of two arguments

! FACT
-factorial (integer argument) / natural log of gamma function (real argument)

SIN COS TAN ASIN ACOS ATAN
-trigonometrical functions (radians)

SIND COSD TAND ASIND ACOSD ATAND
-trigonometrical functions (degrees)

EXP LN SINH COSH TANH
-natural log & hyperbolic functions

LOG ALOG
-base-10 log

DUP DROP SWAP CLEAR
-stack commands (CALC mode only)

Note that starting with version 4.1, Splat! can now correctly handle most negative numbers, such as SIN(-X), whereas in the past this could cause serious problems (since the minus sign looked like subtraction).

With the `/n1:n2` option the data is only modified for points between `n1` and `n2`. By default, the index variable `I` is the absolute index value, i.e. `MATH/4:7 Y=I` will set `Y=4` for point 4. Use the `/J` option for relative indexing, i.e. `MATH/J/4:7 Y=I` will set `Y=1` for point 4 (since this is the first point in the restricted range).

Some examples:

`MATH X=X^2`	$x_i \rightarrow x_i^2$
`MATH Y=X-4`	$y_i = x_i - 4$
`MATH Y=(C2-C3)/C4`	$y_i = (c_2^i - c_3^i) \,/\, c_4^i$
`MATH X=3*I-0.5+RAN`	$x_i = 3i \pm 0.5$
`MATH Y=SIN X`	$y_i = \sin(x_i)$
`MATH Y=EXP( CHS X)`	$y_i = \mathrm{e}^{-x_i}$
`MATH/LINE Y=Y+4*X`	$y_i^{\mathrm{line}} = y_i^{\mathrm{line}} + 4\,x_i^{\mathrm{line}}$

NORM`[/LINE][/n1:n2] [I=pt_num] [var=val]`
With no options, this will renormalize all of the `Y` data points so that the peak `Y` value is equal to one. Optionally a new value other than one can be supplied. By including a variable with the optional value, a different variable can be normalized. To choose the normalization point to be a value different from the peak, use the `I=pt_num` parameter.[2] The same transformation is also performed to the `S` data if in `form 2` or `form 4`. With the `/LINE` option, the line data is normalized. Examples:

`NORM`	set peak `Y` value to 1
`NORM 12`	set peak `Y` value to 12
`NORM Y=12`	set peak `Y` value to 12
`NORM I=30 Y=12`	sets `Y` value for point num 30 to 12

OUT
Equivalent to **WRITE** command.

[2]Sorry, but Splat! doesn't allow you to normalize y to be some value at a specified x-value (e.g. $y(x = 2.5) = 10$). However, this is possible if the x-value of interest corresponds to value in the *xys* data set (x_i) and you know the index-`I` of this point (since you could then use `NORM I=val Y=val`.

POINT`[/R]`
Adds the existing line data to the regular xy data points. This does not clear the line.[3] With the `/R` option the xy data is replaced with the line data. If in `form 7` (xyz mode) the z values are transferred as well.

PORT`[/C,/X] x1,x2`
Transfers data, for a selected x range, from the column data set to the points (`X,Y,S`) data set. Only points with `C1` values in the range `x1` $\leq$ `C1` $\leq$ `x2` are transferred. The number of columns transferred is determined by the format: in form=3, `C1`→`X`, `C2`→`Y`; in form=4, `C1`→`X`, `C2`→`Y`, `C3`→`S`.

In xyz-mode (form=7), `C1`→`X`, `C2`→`Y`, `C3`→`Z`. Assuming $z = f(x, y)$, the `x1,x2` selection limits are only applied to the x-coord, with y allowed to take on any possible value. If you instead want to select a subset of the column data based on the y-coordinate, you need to interchange the roles of columns `C1` and `C2` using **SWAP** (e.g. to select y-values between 0.5 and 2, `SWAP C1,C2`, `PORT 0.5,2`, `SWAP X,Y`). **SWAP** can also be used to re-arrange data in the column array so that the desired column appears in the correct order for the standard column→ xys mapping to work the way you want.

By default, the points are appended to the existing xys data set, to replace the existing data use the `/C` option. The `/X` option replaces the existing data (as with `/C`) and deletes any line data. See also the **ADD** command.

PLOT[/options]
Options include:

`/DATA`	force x&y plot limits to min/max of actual data
`/n1:n2`	plot only points `n1` to `n2` (no expressions allowed)
`/NSET1=n1,n2,...`	plot first `n1` points as squares, then all points up to `n2` as 'xs, *etc.*...
`/NLINE=n1,n2,...`	starts a new line after `n1`, `n2`,...

[3]Note that the corresponding point → line function **LINE** does set the number of data points to zero. Hence, if you start off with xy data points, and then issue `LINE` followed by `POINT` you end up with the same data in both the xys data array and the xy line data array with `N`=$\mathtt{N}_{\mathrm{line}}$.

/XLOG	plot x-axis on a log scale
/YLOG	plot y-axis on a log scale
/ZLOG	plot z-axis on a log scale [3D plotting]
/CONTOUR	2D contour plot of (xyz) data
/DROP	add drop lines to data points (and line segments in 3D plots)
/THETA=value	rotation angle θ (in degrees) of coordinate axes [3D plotting]
/XNM	plots data in color assuming x-coordinate is wavelength in nm (also /YNM, /ZNM)
/XA	plots data in color assuming x-coordinate is wavelength in Å (also /YA, /ZA)
/XM *or* /XU	plots data in color assuming x-coordinate is wavelength in μm (also /YM, *etc.*...)
/XS, /YS, /ZS, /IS	color code data based on value of X,Y,Z, or index-I (blue=small values → red=large values)
/LEG[=position]	draw a plot legend in upper left corner (or other position)
/TEXT	text mode output (retro Splat!)
/PRINT	produces HPGL output file (see below)
/PS	produces PS output file
/BLACK	black background on PS output [/PS only]
/OVERWRITE	overwrite existing HPGL/PS output file (for use with /PRINT or /PS)
/SIZE=size	change plot size (see below)
/MODE=video_mode	change the graphics mode (see below)

Despite the bewildering variety of options, simply typing PLOT is usually sufficient to produce a simple 2D plot of all data (points and lines). In form=7 (xyz) the same PLOT command produces a 3D plot. **PLOT** will normally choose appropriate x&y data ranges (from both the line and point data), and plots the data points as squares. If there are a large number of data points, the size of each symbol is automatically reduced. The /DATA switch forces the plot limits to the actual data limits instead of using round numbers. To plot data on a base-ten log scale, use the /XLOG and/or /YLOG options– note that only positive values are allowed for a log scale. The /DATA switch is ignored on any axis plotted on a log scale.

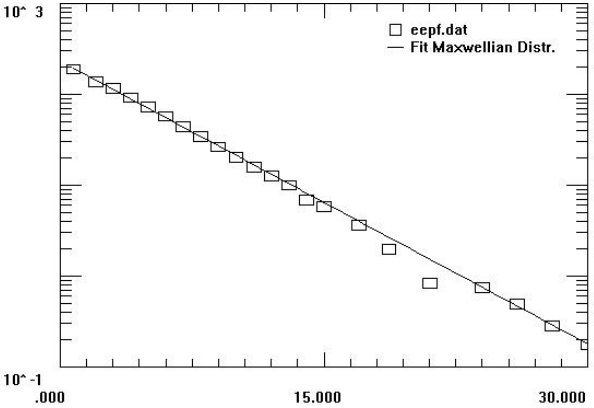


Figure 3.2: Sample log plot, `PLOT/YLOG/LEG=2`.

If in form 4 (xys) the `S` value is used to create an error bar on the `Y` value. To remove the error bars, you must use **FORM** to switch to form 3.

Different sets of data can be plotted with separate symbols using the `/NSET1` and `/NLINE` options. With the `/NSET1=n` option, only the first `n` points are plotted as squares, remaining points as Xs, *etc.*... To have all your data points be Xs, use `/NSET1=0`. With the `I_MARKER` option in the `SPLAT.INI` file (see section 4.5), Splat! can be configured to automatically append `/NSET1` and `/NLINE` options onto each **PLOT** command.

Splat! normally color codes data points and lines using the marker information provided in the in `/NSET1` and `/NLINE` options. Alternatively, plots can be color-coded assuming that a specified variable corresponds to the point's wavelength. For example, the `/XNM` option causes Splat! to color code points and line segments assuming the x-coordinate corresponds to a wavelength measured in nanometers (see sample plot). x/λ-values outside the visible spectral range ($\sim$400-700 nm) are plotted as dark violet (short λ) or dark red (long λ). Similarly, the `/XA` option works the for wavelengths measured in Angstroms (Å), and `/XU` or `/XM` can be used for wavelengths measured in microns (μm). Colors are approximate, with the maximum number of different colors displayed limited by the color depth of the plot mode (see chart on p. 45 for DOS plot

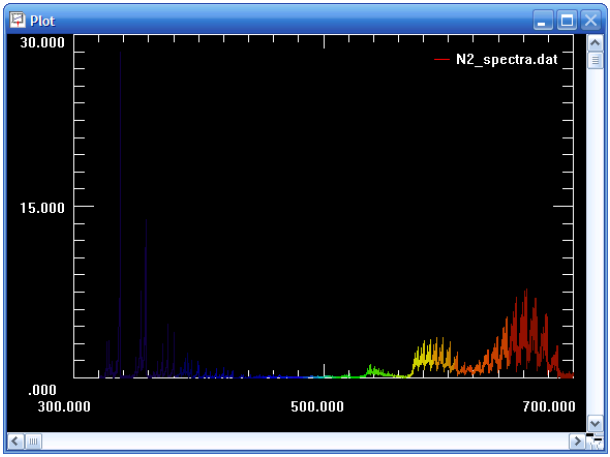


Figure 3.3: Sample of wavelength color coding of N_2 emission spectrum, `PLOT/XNM/LEG=2`.

modes). Plots can also be color coded with the blue-to-red spectral range stretched to cover the minimum-to-maximum range of the selected axis using the `/XS`, `/YS`, and `/ZS` options. Likewise, `/IS` color codes points(or line segments) based on the index number of the point (or line segment).

To include a legend in the upper left corner of the plot, use the `/LEGEND` option. If this overwrites some data points, you can select another location by using `/LEG=position` instead, where position=1 is the upper left corner, 2 is the upper right corner, 3 is the lower left corner, and 4 is the lower right corner.

The `/TEXT`, `/PRINT`, `/PS`, and `/MODE` qualifiers select what form of output device is used. `/TEXT` uses the original Splat! version 1.0 character cell plotting routine. Note that this text output routine does not support many of the more advanced **PLOT** options. `/PS` produces a Postscript (`.PS`) output file called `SPLATn.PS` (`n`=1,2,3...). With the `/OVERWRITE` option Splat! overwrites any preexisting `SPLAT1.PS` file. With the `/B` option Splat! uses a black bacground in the PS output that more closely mimics the screen output.

In form=7 (xyz-mode), `PLOT` produces a 3D plot of $Z(xy)$. The

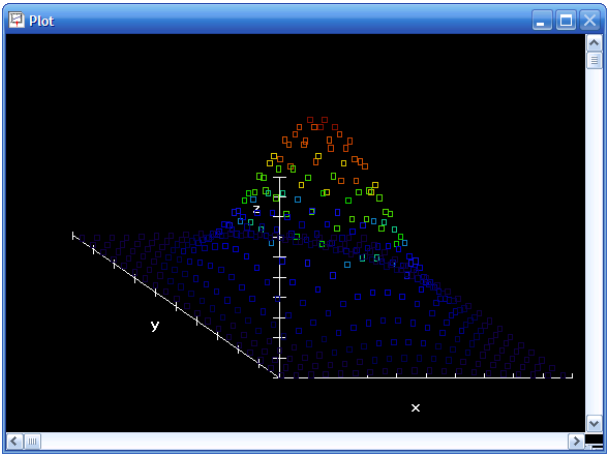
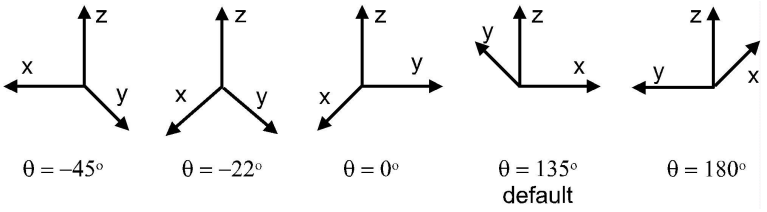


Figure 3.4: Sample 3D data plotted via `PLOT/ZS`.

ranges of the x-,y-, and z-axes are stretched to cover approximately equal lengths in the 2D representation of the 3D data (i.e. the x-range of the data appears to be the same length as the y-range). The `/D` option turns this off, so the y-axis will appear twice as long as the x-axis if the y-data range is twice as large as the x-range. When possible, Splat! only plots positive axes, originating at the (0,0,0) origin. If the origin falls outside the data range, Splat! draws the axes at the minimum value. Droplines produced by the `/DROP` option extend to either the z=0 plane or the $z_{\rm min}$ plane depending on whether the z=0 plane falls within the z-range of the data. The `/THETA=angle` selects the rotation angle (measured in degrees) of the xyz coordinate system.



Using the `/CONT` option (form=7, xyz-mode only), the xyz data is displayed as a color-coded top-down 2D contour plot in a standard

Splat! plot, with the color of the point/line segment corresponding to the z-value.

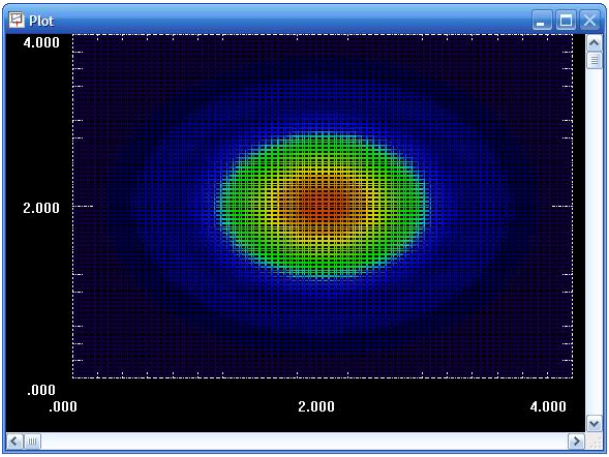


Figure 3.5: Sample 3D data plotted as contour plot, `PLOT/CONT`.

System specific notes: DOS version
`/MODE` is only applicable to the DOS/PC version of Splat!. It selects one of the standard VGA/SVGA graphics modes:

Mode	screen size	colors
`/M=VGA`	640 × 480	16 color
`/M=640`	640 × 480	16 color
`/M=800`	800 × 600	16 color
`/M=801`	800 × 600	256 color
`/M=SVGA`	1024 × 768	16 color
`/M=1024`	1024 × 768	256 color
`/M=1280`	1280 × 1024	16 color
`/M=AUTO`	best CGA, EGA or VGA available	

`/PRINT` produces a HPGL output file called `SPLATn.HPG` (`n`=1,2,3...), and then issues the command `HPSPLAT SPLATn.HPG`. Typically, this is a HPGL file viewer. For actual hardcopy output, this file can be imported into a 'real' program such as most popular graphics and word processing programs. With the `/OVERWRITE` option Splat! overwrites any preexisting `SPLAT1.HPG` file.

System specific notes: Linux version
`/PRINT` produces a HPGL output file called `SPLATn.HPG`, and then issues the command:

```
HP_PRINT1 SPLATn.HPG HP_PRINT2
```

where the `HP_PRINTn`s are defined in the `.splat.ini` file. Typically, this is a HPGL file viewer, such as `hp2xx`. Under Linux, the HPGL file output is the default output mode as well, but Splat! issue the command `HP_PLOT SPLAT1.HPG` to view the output. In addition, in plot (versus print mode) the `/OVERWRITE` option is used to prevent your working directory from filling up with `SPLATn.HPG` files. See section 4.5 for more details on editing the `.splat.ini` file.

System specific notes: WINDOWS menu-version
Two output modes can be used in the Windows version of Splat!. A standard HPGL graphics file can be created with the `/PRINT` option. For actual hardcopy output, this file (called `SPLATn.HPG`, with `n` as a running index) can be easily imported into most popular vector-based graphics programs and many word processing programs. Alternatively, you can use the Windows <alt>-<prt screen> to copy the current Splat! window to the clipboard. This can then be pasted into Paint as a standard Windows Bitmap file.

For the 'conventional' pull-down, menu-driven Windows version, you can use the <TAB> key to switch between the plot window and the command console being in focus. The **LEGEND** command is used to create a separate plot legend in a new window, instead of placing the legend in the plot. By default, the size of the plot window corresponds to approximately 80×24 characters, with the size of the axes filling the window. If you manually resize the window (making it either bigger or smaller), nothing will happen

until the next **PLOT** command, after which Splat! will maximize the plot to fit the new window size.[4]

System specific notes: WINDOWS full-screen-version
For the Windows full screen version, a **PLOT** command clears the screen, and produces a plot in the upper left of the screen approximately 80×24 characters wide (default `size=1`). To create larger (or smaller) plots, use the `/SIZE=size` option to set the magnification factor of the plot size. A value of `/SIZE=2` usually produces an output close to the full screen size. As with the menu-driven Windows version, <alt>-<prt screen> can be used to copy the current Splat! window to the clipboard as a Bitmap image.

QUIT
Exit program.

READ[`/FORM=num,/format_statement`] `filename`
Read in data set from a file. The file can include blank lines. Lines that begin with non-numeric characters are skipped. If you enter '`?`' for the filename, a directory listing is provided before the further prompting for a filename. Splat! also supports wildcards (* and ?) in the filename.

The number of columns of data read in depends on the present data format (see **FORM**). The format can also be set by appending a format qualifier of the type `/FORM=num` or `/num` to the **READ** command.

If in `form=0` (user defined format), a `format_statement` is also required. The format statement is of the type `/'0,0,X,Y'`. Where the format statement consists of a line telling which column in the data file to interpret as each variable (`X,Y,S`), and which column entries to ignore (`0`). The entire statement must be enclosed in single quotation marks. In the preceding example, the numbers in column three will be read into the `X` vector, and the numbers in column four will be read into the `Y` vector. Note that you are still limited to only three total values per line that can be read in (`X,Y,S`). Splat! will switch to the appropriate format after it has read in the data. Some examples of the user defined format

[4]Prior to Splat!'s Win 3.0 version, this didn't work as the plot window was created from scratch for each call. For non-overlapping plot and command console windows, try the Tile command under the Window menu.

(assuming you start in `Form=0`),

`READ/'0,0,0,Y,X' filename`	Reads the 5^{th} col as `X`, 4^{th} col as `Y`
`READ/'000YX' filename`	same as above
`READ/',,,Y,X' filename`	same as above
`READ/0/'0,S,Y,X' filename`	Reads the 2^{nd} col as `S`, 4^{th} col as `X`, 3^{rd} col as `Y`

In `form=5` (auto format), Splat! can read in up to nine columns of numbers. Splat! will find the first row of numbers in the first file it opens, and read in that number of columns for each data point. This data is stored in the `C1` to `C9` data array. The number of points in the `C1` to `C9` array is called `M`, while the number of data columns used is called `C`. If you have no existing *xys* data when you read in with auto format, then Splat! copies as many columns as possible (1 to 3) into *xys* vectors, and sets the format to the appropriate value. For example, suppose you have a file with seven columns of data called `lots.dat`. The command `READ/5 lots.dat` will cause Splat! to read the seven columns of data into `C1` through `C7`. Then Splat! will copy `C1` into `X`, `C2` into `Y` and `C3` into `S`. Finally, Splat! will change the format to `form 4` (*xys* format).

`Form=6` (multi column) is a user unfriendly way to read data into the column data. To use `form 6`, you must also set the number of columns to read with `SET C=n`. All `n` 'columns' need not be on the same line. Splat! will start reading the next line to fill in all the columns.

REGRESS`[/D, /ADD, /NL, /n1:n2]`
Performs a linear regression on the *xy* data (i.e., $y(x) = a + b\,x$), and calculated Pearson's correlation coefficient r.[5] In addition to a standard least squares fit (identical to that obtained from `FIT POLY=1`), a 'robust' fit that minimizes the absolute deviation rather that χ^2 is also calculated [7]. If `form=4`, **REGRESS** also does a weighted least squares fit (the other two fits do not use the *s*-values). By default a 1-segment line for each fit is also created, this can be disabled with the `/NL` option. The default $x_{\min}$ and

[5] Valid r-values range between -1 and $+1$, with a value of zero corresponding to uncorrelated data.

$x_{\max}$ limits for the line segments are 'nice' rounded version of the true x-limits, use the `/D` option to force the limits to that of the data. These lines replace any existing lines unless the `/ADD` option is used.

SET`[/LINE] N=expression`
Set the number of data points (`N`) using some expression. With the `/LINE` option, you can alter the number of line segments. Some examples:

`SET N=12`	set the number of data points to 12 (regardless of current `N`)
`SET N=N+6`	add 6 data points (same as `MAKE 6`)
`SET N=N/2`	cut the number of existing data points in half
`SET N=N*2`	double the number of existing data points
`SET M=8`	set the number of column data points to 8
`SET/LINE N=N/2`	cut the number of existing line segments in half

The expression is evaluated the same as with **CALC** and **MATH**, so you can use all the wacky functions you want. The finally result is rounded to the nearest integer. **SET** is a much more robust version of **MAKE**. `MAKE nsize` is equivalent to `SET N=N+nsize`.

SET is also used to control the column data variables, `M` and `C`. `M` is the number of data points stored in the column array. `C` is the number of columns that contain data. The value `M` can be used in any expression (i.e. with **SET**, **MATH**, the range option `/n1:n2`), but `C` can not.

SHOW
List data on screen. (Same as **LIST**)

SMOOTH[`/G` or `/M`] `window`
Smooths the y data without changing the number of data points. There are two smoothing functions. The default choice (`/G`) is to convolute the data with a Gaussian width FWHM=`window` (i.e. $y_{\rm new}(x) = y(x) * G(x)$). The other choice (`/M`) is to use a rolling median of width equal to `window`. While smooth does not change the number of data points, it does sort the data.

If in xyz-mode (`form 7`), the convolution is applied to $z = f(xy)$ along both the x-axis and y-axis using the same width `window`. If you only want to smooth one coordinate, or if the x- and y-axes have different natural length scales, you need to use **MATH** to rescale the axes before smoothing. For example, suppose the step size between points is 0.1 along the x-axes and 10 along the y-axes and you want to smooth over $\sim$3 points along each axes. This can be done using the following three commands: `MATH Y=Y/100` (to temporarily make $\Delta y \approx 0.1$), `SMOOTH 0.3`, `MATH Y=Y*100` (to restore y-scale).

Splat! trick

If the x-point spacing is irregular, but you always want to smooth over a constant number of points (rather than a constant interval), temporarily replace `X` with `I`. For example, `MATH C9=X` (temporarily store the `X` values in `C9`), `MATH X=I` (label `X` by point number), `SMOOTH 5` (smooth over $\sim$5 points), `MATH X=C9` (restore `X` values).

The number of computer steps required to smooth data scales as `N`^{2}. For large data sets, this can take a long time. To prevent you from thinking Splat! has crashed, a progress bar is displayed.

SORT`[/D] [variable]`
Sorts the data in ascending order either using the default variable, or the optional `variable` parameter. Use the `/D` qualifier to sort in descending order.

STAT`[/n1:n2] [variable]`
Statistics (average, mean, min, max...) on a single variable (either the default variable or the optional `variable`). If in `form 2` or `form 4`, the `S` vector is used to find the weighted mean.

SWAP`[/n1:n2,/LINE] var1=var2`
Interchanges two columns of data specified by `var1` & `var2`. For example, to swap the roles of the x & y vectors, use `SWAP X=Y` or `SWAP X,Y`. `var1` & `var2` can be any of the standard choices of `X`,`Y`,`S` (or `Z`), and `C1-C9`.

WRITE`[/n1:n2,/LINE,/FORM=num,!]  filename`
Writes data out to a file (standard ASCII text file). If the `I_CLOBBER` value in the `SPLAT.INI` file is set to one, the program will check to see if you are overwriting a file. If a file exists, it allows you to abort the write. The `!` option over-rides this over-write protection. Use `/LINE` to write out the line data to a file. The `/FORM=num` option allows you to change the user format before writing.

YO
Hands out a worthless nugget of wisdom. Type `YOYO` for twice the fun, or `YO!` for even more 'advice'.

Chapter 4

Installation & Setup

Splat! comes in multiple varieties: a 16-bit DOS version, two 32-bit Windows versions, and a Linux version. The DOS version works fine with the various Windows operating systems, it just runs in its own DOS/command prompt process. See chart below.

Table 4.1: Compatibility of different Splat! versions with various operating systems.

Operating System	DOS Splat!	Windows Splat! (Full Screen)	Windows Splat! (Menu version)
DOS	Yes	No	No
Windows 95/98	Yes	No	Yes
Windows XP	Yes	Yes	Yes
Windows Vista/7	No	Yes	Yes

Unfortunately there is no handy installation utility for Splat!, so it requires a bit of work to set it up properly. The following subsections describe how to set up each different version of Splat! with different operating systems. The last two sections (4.5 and 4.6) describe how to describe to configure how Splat! operate (common to all versions) using the `splat.ini` initialization/configuration file.

4.1 Windows (32bit) Pull down menu version

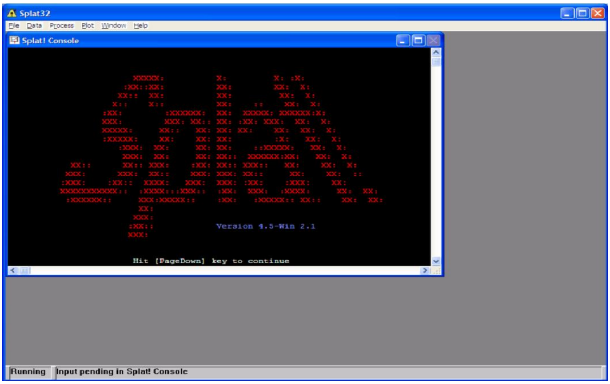


Figure 4.1: Windows XP / 2000 / 95 / 98 / Vista / Windows 7

4.1.1 Installation

The Windows version of Splat! only needs two files:

```
SPLAT.EXE
SPLAT.INI
```

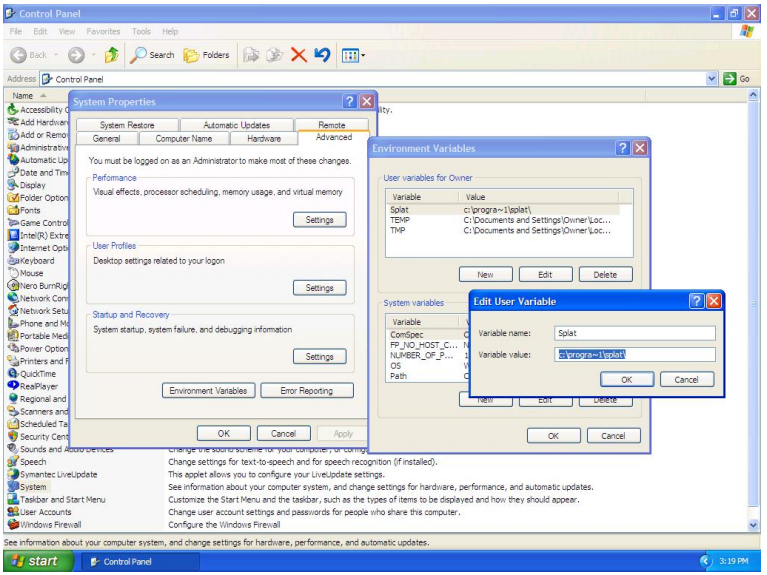
Copy these two files[1] to a directory such as `C:\Program Files\Splat` or `C:\SPLAT`.

The executable should be in a directory listed in your path statement. The `SPLAT.INI` file is optional, but is used to set some common user options. To use the `.INI` file you must also define a `SPLAT` environment variable.

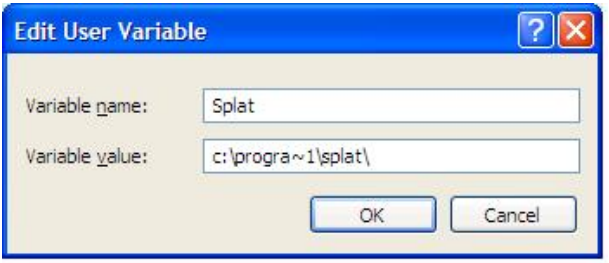
Windows XP/2000/Vista/7 You add the path and environmental variable via the System control panel. Click on the "Advanced" tab, then press the "Environment Variables..." button. While this example is illustrated using Win XP, the Vista and Win7

[1]Sometimes the `.EXE` file is named `SPLAT32.EXE` to distinguish it from the DOS version of Splat!.

versions are fairly similar. For example, to get to the same point in Win7: go to Control Panel > System and Security > System, and then choose Advanced system settings, Environmental Variables.

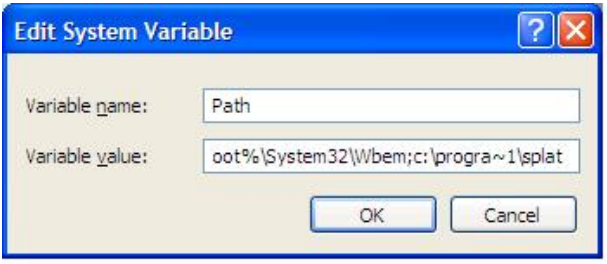


Let us assume you have put the two files in `C:\Program Files\Splat`, in the top User Variables part hit the 'New...' button. , type in `SPLAT` for the name of the new variable and set the value to the Splat! directory. Note that this variable should end with a '\'. It is also required to use the short 8 character path name for the directory, thus for example the value is `C:\Progra~1\Splat\`.



It is also advisable (but not required) to add Splat! to your path. In the System Variables section (the lower portion of the Environmental Variables window), select Path and click the 'Edit...'

button. Append the Splat! directory to the existing Path variable. For the path value, do not include the trailing '\'.



Windows 95/98 To set environmental variables in Windows 9x, you edit the `AUTOEXEC.BAT` file. For example, if Splat! is in the `C:\SPLAT` directory you would have something like:

```
PATH=C:\DOS;C:\WINDOWS;C:\SPLAT
SET SPLAT=C:\SPLAT\
```

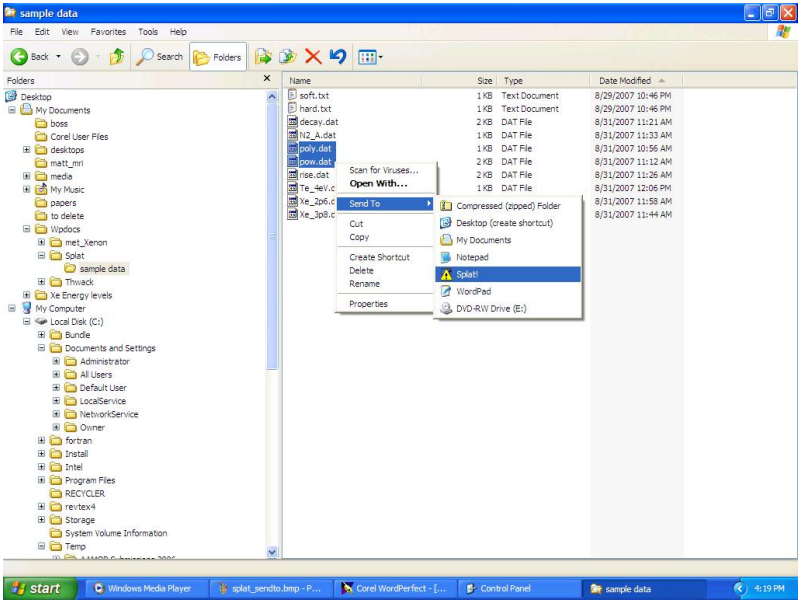
Note the extra '\' in the `SPLAT` variable versus the `PATH` variable.

4.1.2 Running the Program

Splat! can be run just like any other windows program, i.e.: (*i*) enter `Splat` in the Run... box, (*ii*) click on the Splat! program icon, or (*iii*) create a shortcut to the program (by right clicking on the program and selecting create program shortcut), and put a copy on your desktop, or in your Start menu, whatever you want. You can Drag and drop files onto the program icon. While the windows version of Splat! supports long filenames in general, command line arguments (for some reason) get converted to DOS compatible 8.3 style filenames. This applies to drag and drop files. If you create a shortcut, it may be worthwhile to edit its properties and change the 'Start In' directory to something short and sweet like `C:` or `C:\temp`.

One of the most useful ways to run Splat! is to place a Shortcut to Splat! in the Send To folder. In Windows XP (*etc...*) this folder is located somewhere like
`C:\Documents and Settings\User\Send To`.
In Win7, the directory is something like,

`Users\name\AppData\Roaming\Microsoft\Windows\SendTo`.[2] In Windows 95/98 this folder is located in `C:\Windows\Send To`. You can then send files directly to Splat! by right clicking on them in Explorer.



4.1.3 Issues/Bugs

Splat! was written as a command line program in the days of stones knives and bear skins. As a result, trying to run Splat! as a pull-down, menu-driven, Windows program is going to have some problems. Here are a few:

1. **Long Filenames**
   Splat! can read in long filenames of up to 70 characters (including the path). But Splat! can only save files in the "old-school" 8.3 character format. Note that the 70 character limit can cause serious problems with long Windows-XP directories of the form `C:\DOCUMENTS AND SETTINGS\OWNER\MY DOCUMENTS\...`

[2] Note that by default AppData is a hidden directory.

which eats up around 50 of the 70 possible characters. This difficulty can be easily overcome by using directories with short names.

2. **Menu/Keyboard problems**
   The pull down menus in the Windows version look nice, but on some Win-XP systems some menu items don't work (most importantly Open...). Another problem on some systems is that accessing the menus causes the keyboard to lock up. General advice is not to use the menus, you don't really need them any ways.

3. **Plot Window problems**
   When the color depth is too high (millions of colors / high / 32 bits), the windows Splat! interface may not read the display resolution settings correctly, so an empty plot window appears. To fix this, you can either right click on `SPLAT.EXE`, then select Properties, under the compatibility tab check the "Run in 256 color" mode; or you can lower the Color Quality of your display to (medium / 16 bit / thousands of colors). The first option causes weird pallet re-mapping effects, the second lowers the color quality of your display for all programs. I can't tell too big a difference between the medium and high color depths, but you might...

4. `HandleTEMPORARY` **files**
   On Windows 9x systems, `HandleTEMPORARY` files get left behind when Splat! closes. An annoyance, but not a real problem.

4.2 Windows (32bit) Full Screen version

4.2.1 Installation

The full screen Windows version of Splat! also only uses two files:

```
SPLAT.EXE
SPLAT.INI
```

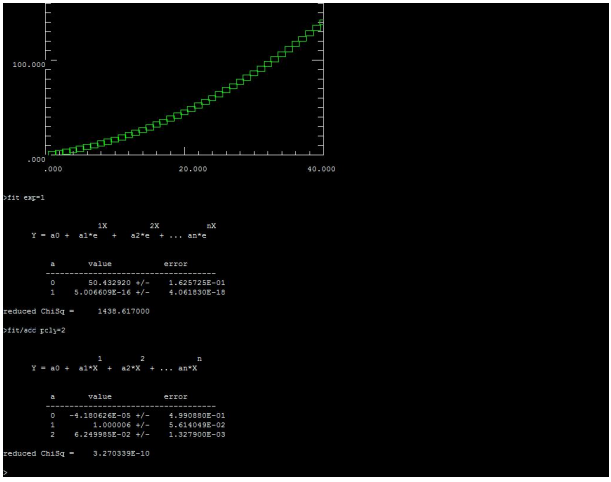


Figure 4.2: Windows XP /Vista /7

Copy these two files[3] to a directory such as `C:\Program Files\Splat` or `C:\SPLAT`, and follow the installation instructions in Sec. 4.1.1. Note, however, that it may not be necessary to define a `SPLAT` environment variable to use an `.INI` file, as this version of Splat! Will also look in the working directory for a version of `Splat.ini`.

Windows XP /Vista/7 Warning

If the color depth is too high (>16 bit in Win XP) or if you are using Win7, the first time you run Splat!, you may not see anything. Or more specifically, you may get a blank screen. Under Win7 the program also crashes, while under WinXP, the program keeps running but you can't see anything (if so, quit the program by typing **EXIT**). This can be fixed by right clicking on `SPLAT.EXE`, choosing Properties, and under the Compatibility tab choosing "Run in 256 colors".

[3]Sometimes the `.EXE` file is named `SPLATx.EXE` to distinguish it from the DOS version of Splat!.

4.2.2 Running the Program

See Sec. 4.1.2 for numerous ways to run Splat! including: the run box, the command line, drag and dropping files, and using the right click Send To option.

While technically a Windows program, the full screen Windows version of Splat! is functionally closer to the DOS command line version. Nevertheless, it does have a few Windows-like capabilities built into it. For example, Notepad is used to **EDIT** files instead of the DOS editor. Also, if **READ** (or **WRITE**) is entered without a filename, if you press <enter> at the `_filename:` prompt, a dialog box will appear allowing you to specify a filename.

4.2.3 Issues with Windows XP/Vista/7

Here are few issues that have arisen with using the Windows full screen version of Splat! with Win XP and Win7. See also issues with long filenames discussed in Sec. 4.1.3.

1. **Blank Screen of Death / Issues with Color Depth**
   When you first start up Splat!, you may see only a blank screen, this is caused by issues with the color depth of the display being more than the primitive graphics drives Splat! uses. See note on page 58 on how to resolve this. Under Windows XP, you can also fix this by using lower the Color Depth/ Quality of your display to (medium / 16 bit / thousands of colors).

2. **Missing text at the ends of lines**
   Depending on the screen resolution, sometimes a few characters will go missing at the end of each line of output and not show up on screen. Annoying, but the characters will reappear as soon as the cursor position reaches the bottom of the screen. As a result, this is most noticeable when either (a) you first start up Splat! (and the cursor is near the top of the screen), or (b) after a **PLOT** command when the cursor is generally about half-way down the screen. By using a larger default plot size (i.e. in the `SPLAT.INI` file set `PLOT_MODE_DEF = '/SIZE=2'`) you can get the cursor to the bottom of the screen a bit faster.

3. **Floating point errors crash the computer**
   While floating point errors (such as dividing by zero) causes all versions of Splat! to crash, when using Windows 7, this may also cause the computer to go into some sort of hibernation/shutdown mode. Splat! attempts to 'trap' floating point exceptions and crash gracefully, but this doesn't always appear to work correctly.[4] Sorry.

4.3 DOS (16bit) version

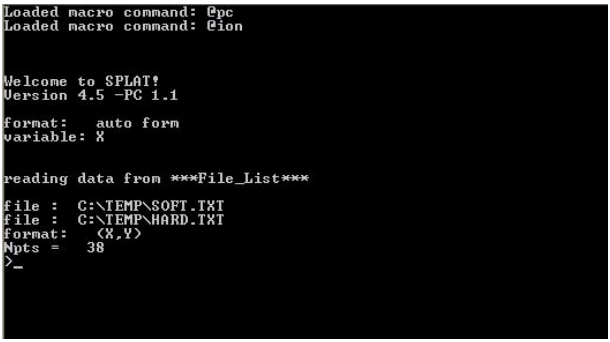


Figure 4.3: DOS / Windows 3.1, 95/95/Me/2000/XP

The DOS version of Splat! might be more properly called Splat! in 'full screen' mode, for DOS and early versions of Windows. As a result, instructions are included for running this version of Splat! in Windows XP (*etc*...) as well as with legacy DOS computers. For later versions of Windows (Vista/7/...), use the 'full screen' Windows version of Splat! as described in Sec. 4.2 .

4.3.1 Installation

The DOS version of Splat! requires the following files:

[4] This may have to do with running multiple programs at once on a dual-core machine. So it is best to be running Splat! alone if you plan on dividing by zero.

```
SPLAT.EXE
DOSXMSF.EXE or DOSXNT.EXE
HPSPLAT.BAT or HPSPLAT.EXE
SPLAT.INI
```

All the executables should be located in a directory listed in your path statement. The `HPSPLAT` files are not required unless you want to automatically process HPGL plot output. The `.INI` file is also optional, but is used to set some common user options and is highly recommended. To use the `.INI` file you must also create a `SPLAT` environment variable. For DOS and early versions of Windows (3.1,95, 98) this is done with settings in the `AUTOEXEC.BAT` file. For example, if Splat! is in the `C:\SPLAT` directory you would have something like:

```
PATH=C:\DOS;C:\WINDOWS;C:\SPLAT
SET SPLAT=C:\SPLAT\
```

Note the extra '\' required at the end of the `SPLAT` variable versus the `PATH` value. In later versions of Windows (XP / 2000) the path and environment variable are setup differently. For these OS follow the directions under the Windows Splat! installation (section 4.1.1).[5]

Configuring the Plot mode

Before using Splat! on a regular basis, it is worthwhile to optimize the screen resolution used for plotting. The default plot mode for Splat! is a very low resolution mode, a much better resolution setting may work better with your computer. To find out, create some sample data in Splat! (or read in the `sample.dat` file included in the distribution file). Now try out various plot modes (see the **PLOT** entry in section 3 for all possible options); i.e. `PLOT`, `PLOT/m=640`, `PLOT/m=1024`,... 'Bad' plot modes manifest themselves in many possible ways: you may see nothing,[6] the screen may freak out, only the text axis labels are displayed, only the data is displayed (with no labels), *etc.*...

Once you find a 'good' plot mode edit the `SPLAT.INI` file to use this choice as the default plot mode. The first few lines of the `SPLAT.INI` file should look something like this,

[5]But note that for the DOS version of Splat! you must add Splat! to your path.

[6]Hitting <return> should fix things up by returning the display to text mode.

```
&DEFAULTS
PLOT_MODE_DEF = '/M=SVGA' ⋮
```

Simply replace the '`M=SVGA`' in the above example with what works best for your computer.

4.3.2 Running the Program

While in DOS the only way to start Splat! is with the command line, there are a number of ways to start up DOS Splat! under Windows as described in Sec. 4.1.2. Before turning to the later, let us briefly examine some of the options of the former.

Splat! allows a few possible command line arguments. In particular, you can set the default form and variable, and specify a filename to read in. For example,

```
SPLAT/FORM=3/VAR=Y STUFF.DAT
```

This starts Splat! up in data format 3 (xy), with `Y` as the default variable, and immediately reads in file `STUFF.DAT`. There is little reason to pre-specify the format since Splat!'s auto-format capability should handle this for you. The filename could include wildcards (*,?), or could be replaced by a list of files separated by spaces. For example, `SPLAT SOME.DAT MORE.DAT` or `SPLAT ARGON_*.DAT`.

To use the DOS version of Splat! with Windows, we first want to create a shortcut to the program. From Explorer right click on the Splat! shortcut and chose properties. Under the Screen Tab, select Full Screen mode. Under the program tab, check the 'close on exit' box. You may want to change the 'Starts in' directory to something simple. It, is also nice to change the Splat! icon. Click the 'change icon...' button, and browse to the Splat! directory. Select the Splat! icon, or find your own icon for Splat!. You should now have a Windows shortcut to Splat!; put a copy on your desktop, in the Send To folder, or in your Start menu, whatever you want (see Sec. 4.1.2 for more detailed instructions).

Note that the DOS version of Splat! uses the DOS 8.3 style `filename.ext` form for filenames, and generally can not read in long filenames. However, by either dragging & dropping files onto the Splat! program icon, or by using the Send To method, you can read in long filenames, but you are still limited to 8.3 filenames on output. Send To can also be used to select multiple files at once to

be read into Splat!, but the number of files is limited by the length of the full path names.

4.3.3 Simultaneous Use of DOS Splat! and Windows Splat!

Can't make up your mind? Do you want to run both the 16bit and 32bit versions of Splat! on the same computer? Or both the menu driven and full screen command line 32bit Windows versions of Splat!? No problem. Set up the DOS (or full screen Windows) version of Splat! as described here. Rename the pull-down menu driven Windows Splat! program something like `WinSplat.exe` or `Splat32.exe` and copy it into the Splat! directory. You could also rename the DOS/full screen version, but you are more likely to access the DOS version directly from the command line where the program name makes a difference versus a Windows shortcut that can be labeled whatever you like. Both versions can use the same `.INI` configuration file.

4.4 Linux version

4.4.1 Installation

The Linux version of Splat! uses the following files:

`splat`	Splat! program file
`hp2xx`	HPGL file viewer program (Optional)
`~.splat.ini`	default program values (Optional)

All the executables should be in a directory listed in your path statement (such as `/usr/bin/`). The `.splat.ini` file is optional, but is used to set some common user options. It is generally a hidden file in your home directory. Typical values for the files contents are:

```
&DEFAULTS
PLOT_MODE_DEF = ' '
IFORM_DEF     = 5
```

```
IVAR_DEF      = 1
I_MARKER      = 1
I_CLOBBER     = 1
HP_PLOT  = 'hp2xx -q'
HP_PRINT1= 'hp2xx -q'
HP_PRINT2= '; ls'
```

The default plot method for the Linux version of Splat! is to create an HPGL output file and then hope the user has some way of displaying it, i.e. `hp2xx` (which you need to supply). Check the `hp2xx` documentation for further help on that program's parameters and qualifiers. Splat!'s printer output also relies on `hp2xx`. The output HPGL filename is sandwiched between the user defined `HP_PRINTn` commands. For example, the following settings produce output on a LaserJet printer:

```
HP_PRINT1 = 'hp2xx -q -c11111 -m pcl -i -
F -f- -O50'
HP_PRINT2 = ' | lpr'
```

For these settings, a `PLOT/PRINT` command corresponds to a Linux command line of

```
hp2xx -q -c11111 -m pcl -i -F -f- -O50
SPLAT1.HPG | lpr
```

Alternatively, the only other plot mode under Linux is the old text-based system (see example p. 20). To set this as the default plot mode set `PLOT_MODE_DEF = '/TEXT'`.

4.4.2 Running the Program

Splat! allows a few possible command line arguments. In particular, you can set the default form and variable, and specify a filename to read in. For example,

```
splat -form=3 -var=Y Stuff.dat
```

This starts Splat! up in data format 3 (xy), with `Y` as the default variable, and immediately reads in file `Stuff.dat`. The filename could include wildcards (*,?), or could be replaced by a list of files separated by spaces. For example,

```
splat -form=3 some.dat more.data
```

4.5 SPLAT.INI Configuration File

The `SPLAT.INI` file contains two things, (1) the default start-up mode information for Splat! and (2) macro definitions. The macro functions are described in section 4.6. As for the default start-up mode stuff, you do not absolutely need an `.INI` file since Splat! has up to three sets of default parameters for most options. These sources are (in order of precedence):

1. Splat!s own internal defaults
2. `SPLAT.INI` values
3. Values supplied on the command line (i.e. `SPLAT/FORM=3`)
4. Splat! Command qualifiers (i.e. `PLOT/M=VGA`, or `FORM 4`)

Nonetheless, the `.INI` file is the only way to turn on/off the marker function (for automatic generation/management of marks and labels), the control of file overwrite protection, and define macros.

The exact contents of the `SPLAT.INI` file depend upon what operating system you are using. The `SPLAT.INI` file distributed with your particular version of Splat! contains lists of all valid values for the setting for your system.

For the DOS/Windows version of Splat!, typical `SPLAT.INI` contents and meanings are:

`&DEFAULTS`	Required header
`PLOT_MODE_DEF='/M=SVGA'`	Sets the default plot mode, see **PLOT**
`IFORM_DEF = 3`	Sets the default data format see **FORM** entry in section 3
`IVAR_DEF = 1`	Sets the default variable see **FORM**
`I_MARKER = 1`	Set to 1 to automatically plot each data file read in with a new symbol, 0 for same symbol
`I_CLOBBER = 1`	Set to 1 to check for overwriting output files, 0 to allow overwriting without checking

The `PLOT_MODE_DEF` option is appended onto every **PLOT** command. It is typically used to set the display resolution for the DOS

version of Splat!. However, any valid **PLOT** options can be specified. For example, if you always want a plot legend, you can set it to `/LEG/M-SVGA`. Or under the full screen Windows version of Splat!, if you want plots to be 75% larger than the default size, set `PLOT_MODE_DEF='/SIZE=1.75'`.

4.6 Macro Commands

Up to ten command macros can be defined in the `SPLAT.INI` file. Each macro can be composed of up to 20 Splat! commands.[7] A macro command is accessed in Splat! by typing

```
@macro_name  [parameter]
```

at the Splat! prompt. All macro names must start with a `@`. One optional parameter is allowed. The macro definition block in the `SPLAT.INI` file is of the following form:

`@macro_name`	the macro name
`3`	number of Splat! statements in the macro
`statement 1`	Splat! commands...
`statement 2`	
`statement 3`	

Anytime a '`%1%`' appears in the macro statements the user supplied parameter is substituted (if provided). For example, suppose you want a macro that creates a ten point (xy) data set for a user supplied function on the interval from 0 to 1. In the `SPLAT.INI` this would be defined as

```
&MACRO
@makeit
5
form 3
set n=10
label user function: %1%
math x=(i-1)/9
math y=%1%
/
```

[7] 20 is more a recommendation than a firm limit. Up to 30 commands *may* be allowed.

To use this macro command to create values for $\sin(x)$ one would enter `@makeit sin(x)` at the Splat! prompt. The sample `SPLAT.INI` file included in the Splat! distribution package contains a slightly more advanced version of this macro that uses relative indexing and adds the 30 point user function to the existing data set (without erasing any existing data as in the previous macro):

```
@makeit
5
form 3
make 30
label user function: %1%
math/j/n-30+1: x=(i-1)/(30-1)
math/j/n-30+1: y=%1%
```

Macro calls can be nested (one macro invokes another macro), but not recursive.

Appendix A

Technical Details

Splat! is written in Fortran 77 (using many DEC VAX extensions).[1] The original version of Splat! ran on a microVAX before being ported to PC computers. Most of the numerical processing routines (sorting, fitting, *etc...*) used by Splat! are adapted from those in *Numerical Recipes* [7]. The equation parsing system used by the **MATH** command is heavily adapted from 'Lexical Analysis' example from Chap. 10 of *FORTRAN 77 for Engineers and Scientists*[6].

A.1 Averaging

To illustrate what is going on with Splat!'s **AVERAGE** command, let us consider the sample data illustrated in Fig. A.1. The plot is a histogram of 279 individual measurements.[2] The average of all the measurements is equal to 17.4. The standard deviation (σ) of the measurements is 10.5. Indeed, if the measurements are averaged (in form=3) with `AVG/SD`, and then listed (after switching to form=4) this is the listed error `S`. Note, however, that by default `AVG` will

[1] With the exception of a few Windows interface routines written in Fortran-90.

[2] In this context, (A-B) is equal to difference in counts that accumulates in two counters (A & B) over 20 seconds. The A counter collects signal while an electron beam simultaneously ionizes/excites Ar ground state atom to the the $Ar^{+}(4d^{2}P_{1/2})$ level, while the B counter accumulates background counts while the electron beam is off. Negative (A-B) counts are possible if the signal counting rate is not much larger than the background counting rate.

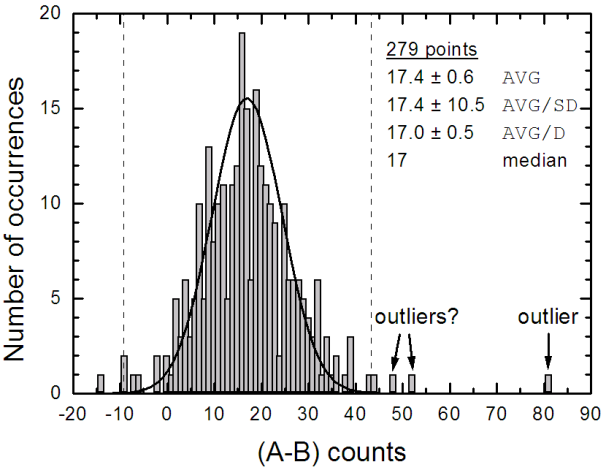


Figure A.1: Histogram of photon counting signal for the $Ar^+(4d\,^2P_{1/2} \rightarrow 4p\,^2P^o_{1/2})$ transition (330.73 nm) measured at an incident electron energy of ∼90 eV. (Raw data used to produce cross section values in Ref. [2].)

produce a much smaller uncertainty value of ±0.6. By default, Splat! lists the *error in the mean* (or standard error) which is equal to $\sigma/\sqrt{N}$ where N is the number of measurements. Considering the large size of N, the $N^{-0.5}$ factor significantly reduces the standard error from the standard deviation.

The standard deviation can be thought of an estimate of the expected variability of the next independent measurement of the quantity of interest. So if there had been a 280^{th} measurement, it most probably would have fallen in the range of 7 to 27. On the other hand, this one additional measurement when added to the previous 279 measurements, would hardly change the mean value of 17.4. In contrast, the error in the mean is the expected uncertainty in the mean value which has accumulated from the 279 measurements. If we repeated a similar set of 279 measurements, the mean of this new set would be expected to be within about ±0.6 of 17.4. By default Splat! assumes the error bar you are interested in is the uncertainty in the mean, rather than the uncertainty expected

in an individual measurement (i.e. the standard deviation).

Being real data taken in a real world, the measured distribution in Fig. A.1 does not exactly match the smooth Gaussian curve obtained from probability theory. While most of the values cluster around the range of 0-40, the single value at 81 stands out as anomalously large. While this single value may be some sort of statistical fluke, it is far more likely the result of an experimental glitch- electrical noise picked up in one counter channel more than the other as someone turned on a piece of heavy machinery in a room nearby.[3] It would be nice to exclude this 'obviously wrong' data point from our statistical analysis of the remaining points. But what about the two points around 50? Or the point at -14? All of these points stand out in comparison to the smooth Gaussian curve, but don't seem that off from nearby points that aren't in question. To take the human judgement/bias out of this process, Splat! includes the built in `/DISCARD` option. With the `/DISCARD` option, Splat! does two rounds of statistical analysis. In the first round it calculates the standard deviation and median value. Since half of the data points have a value larger than the median, and half have a value less than the median, a few outliers/bad points don't shift the median value all that much. As a result, it is a robust indicator of the mean value even in the presence of a few 'bad points'.[4] If a point lies more than 2.5 standard deviations from the median value, it is considered 'bad' and excluded from the second round where the mean and revised standard deviation are calculated for the 'good' points. Without any bad points, this criteria is expected to throw away about 1% of 'good' points that just happen to lie at the edges of the normal distribution. For the example data in Fig. A.1 these limits are shown as the vertical dashed lines at -9 and +43. A total of 5 points are excluded by `AVG/D`, four on the right side of the distribution, and one on the left side. This shifts the mean value down slightly to 17.0 ± 0.5.

For averaging with error bars (form=4, *xys*-mode), Splat! calculates the average via [1]

$$y_{\rm avg} = \frac{\sum_i (y_i/s_i^2)}{\sum_i (1/s_i^2)} \ , \tag{A.1}$$

[3] Stray cosmic rays are also a common excuse.

[4] The *mean* value, however, can be thrown off significantly by a single bad point that differs from the true mean by a large amount. See the example on p. 23.

with the uncertainty in the average value equal to

$$s_y = \sqrt{\frac{1}{\sum_i (1/s_i^2)}} \ . \tag{A.2}$$

Splat! averages y-values for all points at the 'same' x-value. To allow for numerical round off errors, Splat! actually averages all points together within x-axis bins of width δx. Unless specified by the user (via the `AVG delta_x` qualifier), Splat! uses a default value of $\delta x = 0.0001$. The x-value associated with the averaged y values is obtained by taking the average of the x-values within the acceptable window. Note that this procedure can sometimes produce an unexpected result. Imagine you have four points at $x =$ 1.0000, 1.0001, 1.0002, and 2.0000. You might expect to end up with two averaged points at 1.0001 and 2.0000, but Splat! would instead find only the first two points within the default 0.0001 window of the first x-value; and as a result, Splat! would produced three averaged values at $x = 1.00005$, 1.0002, and 2.0000. To overcome this difficult, for this particular pathological case, one could either use something like `AVG 0.1`, or `MATH X=NINT(X)` and then `AVG`. While both methods would produce two averaged xy values, the x-coordinates for the first point would be slightly different in the two cases: 1.0001 for the former, 1.0000 for the latter.

Table A.1: Effect of **AVG** command under different circumstances. The chart does not include the effects of binning or discarding data. In this context, N represent the number of data points with the same x-value that are being averaged together. Note: $\langle y \rangle = N^{-1} \sum_i y_i$, and $\sigma_y^2 = (N-1)^{-1} \sum_i (y_i - \langle y \rangle)^2$.

Format	command	`X`	`Y`	`S` or `Z`
(xy) form=3	`AVG`	x	$<y>$	$\sigma_y/\sqrt{N}$
(xy) form=3	`AVG/SD`	x	$<y>$	σ_y
(xys) form=4	`AVG`	x	$\frac{\sum_i (y_i/s_i^2)}{\sum_i (1/s_i^2)}$	$\left[\sum_i (1/s_i^2)\right]^{-1/2}$
(xyz) form=7	`AVG`	x	y	$<z>$

A.2 Smoothing

Splat! can smooth either xy data or xyz data using one of two smoothing options: a rolling median or a Gaussian window. Note that for smoothing purposes xys data is treated the same as xy data as the error bars (s) are neglected in the smoothing process. Associated with each of the smooth functions is a scale length Δ (either the width of a sliding window or a Gaussian FWHM). For xy data this scale length is compared with the x-distance between each pair of data points via $|x_i - x_j|$. For xys data the scale length is compared to $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. The two smoothing functions are illustrated using the xy scheme, but are the essentially the same for xyz data with y replaced with z and $|x_i - x_j|$ replaced with $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Rolling Median For each point i, create the set Y_i^C which consists of the y_j-values of all points with $|x_i - x_j| \leq \Delta$. Then $y_i^{\rm sm} = \mathrm{median}(Y_i^C)$.

Gaussian Converting the supplied Gaussian FWHM into an exponential scale factor aa,

$$aa = \frac{\Delta^2}{4\ln(2)} \; . \tag{A.3}$$

We compute,

$$y_i^{\rm sm} = \frac{\sum_j \left(y_j \; e^{-(x_i - x_j)^2/aa} \right)}{\sum_j \left(e^{-(x_i - x_j)^2/aa} \right)} \; . \tag{A.4}$$

As a last step in both cases, Splat! copies $y_i^{\rm sm}$ into y_i.

A.3 Fitting

Fitting entails minimizing the difference between a set of experimental $y(x)$ values and those derived from a trial function $y_{\rm fit}(x; a_1, a_2, \ldots)$ by varying the fitting parameters $a_1, a_2, \ldots$ [1, 7]. The 'difference' between the experimental values and the trial values is defined to

be χ^2,

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - y_{\rm fit}(x_i; a_1, a_2, \ldots, a_K)}{\sigma_i} \right]^2 , \qquad \text{(A.5)}$$

where σ_i is the uncertainty in y_i (set to one in xy form=3). This is called the method of least squares fitting. The *number of degrees of freedom* is equal to the number of data points N minus the number of extracted parameters K. To be a well posed problem, you must have at least as many data points as parameters.

The extracted parameters will each have some uncertainty $\sigma(a_k)$ associated with them. Mathematically,

$$\sigma^2(a_k) = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_k}{\partial y_i} \right)^2 . \qquad \text{(A.6)}$$

The details of how this all works can be found in Ref. [7]. Since $\sigma(a_k)$ depends on the uncertainties σ_i in the experimental y_i-values, the $\sigma(a_k)$ error estimates are only as valid as the σ_i error estimates (i.e. you must be in form=4 for them to have 'real' meaning). Technically, the $\sigma^2(a_k)$ values are the *variances* in the set of fitted parameters a_k. There also *covariance* terms that relate the uncertainty in a_1 to a_2 ($etc\ldots$).[5] If you are interested in the full covariance matrix, you are out of luck- Splat! only reports the variance terms.

If the trial function is composed of a series of *basis functions* $X_k(x)$ such that it can be written as a linear combination of the basis functions,

$$y_{\rm fit}(x; a_1, a_2, \ldots) = \sum_{k=1}^{K} a_k \, X_k(x) , \qquad \text{(A.7)}$$

we have the case of *linear least squares fitting.* Barring some numerical processing details, a linear least squares fit always works, i.e. it provides the unique set of parameters $a_1, a_2, \ldots, a_K$ that have the global minimum of χ^2. If the fitting function is not a linear function of the parameters a_k, the method of *non-linear least squares*

[5]For example, consider fitting a straight line $y = ax + b$ to a set of data. For the minimum slope within the uncertainty range, $a - \sigma_a$, the fit can be somewhat improved by increasing the offset b to $b + \delta b$. This change in b with a variation in a results in a covariance (cross talk) between the values of the slope and the offset.

fitting is used. This method has two serious drawbacks: first, the final solution is not guaranteed to be the global minimum of χ^2, but only a local minimum. Second, to better our odds of finding the global minimum we need fairly good starting guesses for the values of the parameters a_k.

As far as using Splat's **FIT** function, you don't see a difference between linear and non-linear fitting. For non-linear fits, Splat! figures out the initial guessing of the parameters. unfortunately, these guesses don't always lead to the global minimum in χ^2. Sometimes Splat! fails to find a good fit because of the 'rules' it uses to guess initial parameters don't work well with your particular data set. In these cases, you may be able to nudge Splat! along by tweaking your xy data to better match its expectations. Here are Splat!'s rules for initial guesses.

Exponential rise		$y = a_0 \left[1 - \mathrm{e}^{-\frac{(x-a_1)}{a_2}} \right]$
a_0 :	aysmtopic val.	y-value of last point, y_N
a_1 :	offset	x-value of first point x_1
a_2 :	rise-time	first x-value that exceeds $(a_0/2)$, minus the x-offset a_1

Gaussian peak		$y = a_0 \, \mathrm{e}^{-(x-a_1)^2 \, / \, a_2^2}$
a_0 :	peak height	maximum y-value $(y_{\max})$
a_1 :	peak location	x-value corresponding to $y_{\max}$
a_2 :	peak width	$2(a_1 - x_{mp})$, where x_{mp} is the first point where the y-value exceeds $(a_0/2)$

Gaussian with bkg.		$y = a_0 \, \mathrm{e}^{-(x-a_1)^2 \, / \, a_2^2} + a_3$
a_0 :	peak height	maximum y-value $(y_{\max})$
a_1 :	peak location	x-value corresponding to $y_{\max}$
a_2 :	peak width	$2(a_1 - x_{mp})$, where x_{mp} is the first point where the y-value exceeds $(a_0/2)$
a_3 :	background	averg. of first & last y-values, $(y_1 + y_N)/2$

Asymmetric Gaussian with background		$y = a_0\,\mathrm{e}^{-\left[\left(\frac{x-a_1}{a_2}\right)^2\left(\frac{1+\exp[a_4(x-a_1)]}{2}\right)^2\right]} + a_3$
a_0 :	peak height	maximum y-value ($y_{\max}$)
a_1 :	peak location	x-value corresponding to $y_{\max}$
a_2 :	peak width	$2(a_1 - x_{mp})$, where x_{mp} is the first point where the y-value exceeds $(a_0/2)$
a_3 :	background	averg. of first & last y-values, $(y_1+y_N)/2$
a_4 :	asymmetry	0. (assume symmetric Gaussian)

Morse Potential		$y = a_0\,\left[1 - \mathrm{e}^{-a_1\,(x-a_2)}\right]^2 + a_3$
a_0 :	dissoc. energy	difference of the maximum & minimum y-values ($y_{\max} - y_{\min}$)
a_1 :	β	fixed value of 1.5
a_2 :	equilib. radius	x-value corresponding to $y_{\min}$
a_3 :	offset energy	minimum y-value ($y_{\min}$)

Maxwell-Boltzmann		$y = \frac{2}{\sqrt{\pi}}\,a_0\,\sqrt{x}\,\frac{1}{a_1^{3/2}}\,\mathrm{e}^{-x/a_1}$
a_0 :	density	$4\times\, y_{\max} \times\, x$-value corresponding to $y_{\max}$
a_1 :	temperature	$2\times\, x$-value corresponding to $y_{\max}$

Druyvesteyn		$y = \frac{1}{\sqrt{\pi}}\,a_0\,\sqrt{x}\,\frac{1}{a_1^{3/2}}\,\mathrm{e}^{-0.243\,x^2/a_1^2}$
a_0 :	density	$2\times\, y_{\max} \times\, x$-value corresponding to $y_{\max}$
a_1 :	temperature	x-value corresponding to $y_{\max}$

Arrhenius Form		$y = a_0\,x^{a_1}\,\mathrm{e}^{\frac{-a_2}{x}}$
a_0 :	amplitude	maximum y-value, $y_{\max}$
a_1 :	power	fixed value of 0.5
a_2 :	act. energy	average of minimum & maximum x-values, $(x_{\max} + x_{\max})/2$

While the exponential decay function,

$$y = a_0\,\mathrm{e}^{a_1\,x}, \tag{A.8}$$

has a non-linear dependence on a_1, it can nonetheless be fit as a

linear equation. By taking the natural log of both side, we obtain

$$\ln y = \ln a_0 + a_1 \, x, \tag{A.9}$$

which Splat! fits as first-order polynomial. While not a standard Splat! fitting function, a similar trick can be used by the user to fit $y = a_0 \, x^{a_1}$. Namely `MATH Y=LN Y`, `MATH X=LN X`, and then `FIT POLY=1`, with $a_0^{\rm true} = \exp(a_0^{\rm fit})$.

A.4 Calculus

Most numerical analysis books have a great deal to say about various numerical methods of evaluating either the integral or derivative of a table of values of $y = f(x)$. Unfortunately, these methods generally assume you can evaluate $f(x)$ at any desired x. As a result, the method generally assume the table of values consist of a set of equally spaced x-values separated by a controllable spacing h. This is in sharp contrast to what Splat! gets supplied with by the user, an arbitrary list of xy values which may contain repeated values and have large experimental uncertainties. As a result, the routines used by Splat! are probably not the best[6], but work well enough given the circumstances.

A.4.1 Integration

Splat! first sorts the data by x, and averages all y-values at the same x-value. This is the unweighted average even in form=4 (i.e. error bars are ignored). Assuming there are K unique x values,

$$\int y(x) \, dx \;\approx\; \sum_{k}^{K-1} \; \frac{x_{k+1} - x_k}{2} \, [y_k + y_{k+1}] \; . \tag{A.10}$$

This is integration by the trapezoid rule (see for example Refs. [4, 3]). It is one of the least efficient/accurate methods of calculating integrals for a fixed lattice spacing. But for potentially unequally spaced points, and with no knowledge of the derivatives of $y = f(x)$

[6] Indeed, if you have equally spaced values for an analytically known function you are MUCH better off evaluating your integral/derivatives by some other means

this is about the best Splat! can do. The error in each portion of the summation is equal to [3]

$$E_k = -\frac{f''(\eta)\ (x_{k+1} - x_k)^3}{12}\,, \quad \text{for some } \ \eta \in (x_k, x_{k+1})\,. \qquad \text{(A.11)}$$

In contrast, integration by Simpson's rule gives an error on the order of $(x_{k+1} - x_k)^5$ [3, 4] which is *significantly* better for small spacing.

A.4.2 Derivation

In comparison to the 'sort-of' suitability of textbook techniques for numerical integration (a.k.a. numerical quadrature), the standard textbook methods for obtaining numerical derivatives (numerical differentiation) are almost completely worthless when applied to arbitrary data sets. The act of integration tends to average out experimental noise in the data: the contribution from one interval where y_k is too high, will be at least partially offset by another interval in the summation where $y_{k'}$ is too low. This offsetting doesn't occur in numerical differentiation, if y_k is too large (by $+s_k$), the derivative for the interval immediately before x_k will be too big, and the derivative for the interval immediately after x_k will be too small. Both are wrong, period. Since $dy/dx \approx (y_{k+1} - y_k)/(x_{k+1} - x_k)$, the smaller the x-spacing the larger the noise magnification from statistical variations in y. Even for analytical functions, numerical roundoff errors lead to instabilities in the calculation of derivatives as the step size is decreased [3, 4]. Things are even worse for the second derivative.

If you have an analytical function, you are MUCH better off evaluating the derivative analytically, or by a method other than Splat!. With this caveat/warning aside, Splat! calculates the derivative at point x_i by fitting a second order polynomial to the y-data in the interval around x_i, and extracting the derivative from the slope of this fit,

$$y^{\rm fit}(x)\Big|_{x_i} \ \approx\ a + b\,x + c\,x^2 \qquad \text{(A.12)}$$

so

$$\frac{dy}{dx}(x_i) \ \approx\ \left.\frac{dy^{\rm fit}}{dx}\right|_{x_i} = b + 2\,c\,x_i\,. \qquad \text{(A.13)}$$

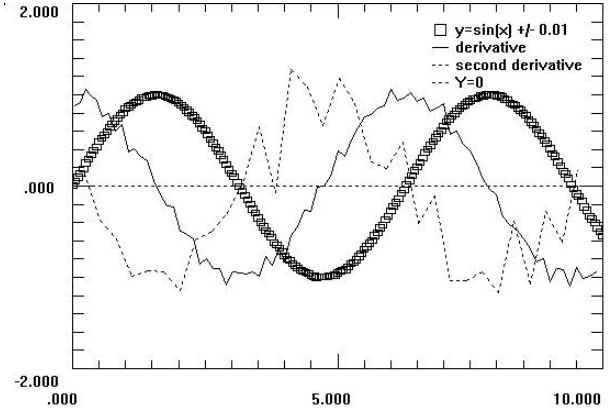


Figure A.2: Sample derivatives of 'noisy' data, $y = \sin(x) \pm 0.01$. Compare to noiseless data in Fig. 3.1 on p. 24. Note how the noise amplitude increases as one goes from $f(x) \rightarrow f'(x) \rightarrow f''(x)$.

The fitting process should help smooth out the random statistical uncertainties in y. In form=4 (xys-mode) this fitting process includes the known uncertainties (s) in the y-values.

In principle, this fitting procedure can be done for an interval around each point (similar to the **SMOOTH** function), yielding a unique derivative value at each x-value. I find this somewhat intellectually dishonest (see note about smoothing on p. 20); if the fit is over an interval of Δx, you should end up with a function with a spacing of about Δx. Unless specified by the user, Splat! sets the default value of Δx at 1% of $(x_{\max} - x_{\min})$. However, since three points are required to fit a second order polynomial, Splat! will increase effective Δx range to include at least three points.

Splat! obtains the second derivative of a function by successively taking two first derivatives. Namely, $y \rightarrow dy/dx \rightarrow d/dx(dy/dx)$. Since Splat! reduces the number of data points with each derivative, the second derivative dataset typically has less than (1/9) the number of starting points.

A.5 Plotting

A.5.1 Nice Numbers

When plotting, the limits of a plot (i.e. $x_{\rm min}$ and $x_{\rm max}$) are not usually the minimum and maximum of the dataset, but 'nice' round numbers. For example, if the data runs from 0.3 to 8.5, the more natural limits for the plot are probably something like 0 to 9 or 0 to 10. Commercial-quality plotting programs allow you to fine tune whatever the program guesses these limits should be, but Splat!'s simplicity prevents you from adjusting these limits (other than setting them to the limits of the data with the `/D` option). Splat!'s method is as follows,

$$P = 10^{{\rm Int}_d[\log(x_{\rm max}^{\rm data} - x_{\rm min}^{\rm data})]} \; , \tag{A.14}$$

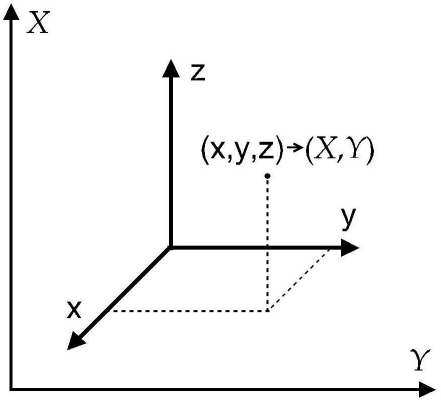
where ${\rm Int}_d(x)$ rounds *down* to the preceding lower integer (i.e. ${\rm Int}_d(1.7)$=1, ${\rm Int}_d$(-1.7)=-2). P represent how many orders of magnitude are cover by the data range. The plot limits are then chosen to be,

$$x_{\rm min}^{\rm plot} = {\rm Int}_d(x_{\rm min}^{\rm data}/P)\; P \tag{A.15}$$

$$x_{\rm max}^{\rm plot} = {\rm Int}_d(x_{\rm max}^{\rm data}/P + 0.99999)\; P \; . \tag{A.16}$$

To consider a couple of examples, for data ranging from 0.3 to 8.5, $P = 1$, $x_{\rm min}^{\rm plot} = 0$, and $x_{\rm max}^{\rm plot} = 9$. For a data range of 1.1 to 1.6, P=0.1, $x_{\rm min}^{\rm plot} = 1.1$ and $x_{\rm max}^{\rm plot} = 1.6$.

A.5.2 3D to 2D coordinate transform



To convert a 3D xyz coordinate to a 2D $\mathcal{XY}$ coordinate to be plotted, Splat! performs the following transformation:

$$\mathcal{X} = s_x\, x\, \cos(\theta - 135^\circ) + s_y\, y\, \cos\theta \quad \text{(A.17)}$$

$$\mathcal{Y} = s_x\, x\, \sin(\theta - 135^\circ) + s_y\, y\, \sin\theta + s_z\, z\,, \quad \text{(A.18)}$$

where θ is the rotation angle of the xyz coordinate system about the z-axis ($\theta = 0^\circ$ in the figure), and s_x, s_y, and s_z are scale factors. Setting $s_x = 1$, the scale factors for the y- and z-axes are determined by

$$s_y = (y_{\max} - y_{\min})/(x_{\max} - x_{\min}) \quad \text{(A.19)}$$

$$s_z = (z_{\max} - z_{\min})/(x_{\max} - x_{\min})\,. \quad \text{(A.20)}$$

If s_y is within $\pm 30\%$ of 1, or s_z is between 0.5 and 2, they are set to one. Likewise, if the `/D` option is used, the scale factors are set to one. For a log axis, the specified coordinate (i.e. z) is replaced with $\log z$ before the above transformation.

A.5.3 $\lambda \rightarrow$rgb conversion

Wavelength color coding is actually rather hard. Or more correctly, accurate color coding is hard, or pretty much impossible. A monochromatic light specified by a given wavelength is a 'pure' color, and

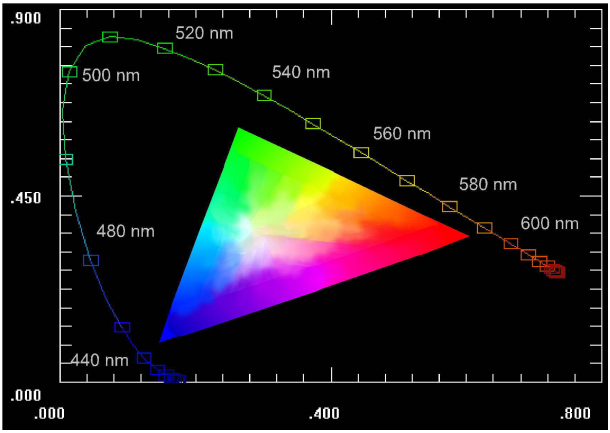


Figure A.3: A poor attempt at a 1931 CIE Chromaticity diagram with color range of a typical computer monitor.

can not be truly reproduced by a computer monitor. To see why, we turn to something called a a 1931 CIE chromaticity diagram.

In this representation, the wavelengths of light make up the outer arc in Fig. A.3. The three corners of the inner triangle in the figure correspond to the red-green-blue (RGB) colors generated by the three phosphors in a display. The color of an individual pixel is described by the specifying the intensity of each of these three colors. For example, assuming a maximum intensity of 255, the color red corresponds to an RGB value of (255,0,0); yellow, which is an equal mix of red and green, is specified as (255,255,0). An equal mix of all three colors produces white. Thus, by mixing the ratios of the three colors, a computer display can represent any color *within* the color triangle of Fig. A.3. Since the wavelength spectrum falls outside of this triangle, a computer can never truly display a wavelength of a monochromatic light source.

Things are further complicated by the fact that the human eye doesn't work like a spectrometer, but has three different photoreceptor cone cells. Each cone cell type has a different photopsin protein that absorbs light in different parts of the spectrum. One type is most sensitive to blue light, one to green and one to red. Thus, the human eye converts a monochromatic light signal into a

weighted rgb signal that our brain interprets as a given color. If the spectral output of the screen RGB phosphors exactly matched the rgb response of the human eye, the color rendering of spectral colors could be easy, but alas this is not the case.

Splat!'s spectral color mapping works as follows. First, the wavelength λ (measured in nm) is mapped to a set of relative RGB values using,

$$R = 0.20 \exp\left[-\frac{(\lambda - 447)^2}{28.1^2}\right] + 0.57 \exp\left[-\frac{(\lambda - 596)^2}{47.4^2}\right] \quad \text{(A.21)}$$

$$G = 0.55 \exp\left[-\frac{(\lambda - 559)^2}{59.2^2}\right] \quad \text{(A.22)}$$

$$B = 0.48 \exp\left[-\frac{(\lambda - 440)^2}{20.0^2}\right] + 0.65 \exp\left[-\frac{(\lambda - 460)^2}{34.0^2}\right] \quad \text{(A.23)}$$

These values are loosely based on the CIE color matching values. Splat! then renormalizes these values relative to the maximum of the three coordinates,

$$M = \mathrm{Max}(R, G, B) \quad \text{(A.24)}$$

$$R_N = R/M \quad \text{(A.25)}$$
$$G_N = G/M \quad \text{(A.26)}$$
$$B_N = B/M\,. \quad \text{(A.27)}$$

This renormalization increases the effective brightness of the color to the maximum allowable amount. Then, the intensity is decreased to mimic the eye's luminous efficiency:

$$C_f(\lambda) = \exp[-(\lambda - 550)^2/220^2]\,. \quad \text{(A.28)}$$

While somewhat based in reality, this function matches neither the photopic (bright light/cone response) or scotopic (dark adapted/rod response) sensitivity curve of the human eye[5]. The width parameter of the curve (220) has been stretched a bit to enhance the response in deep blue-violet and deep red edges of the spectrum. The Windows version of Splat! uses a value (185) a bit closer to reality.

Finally, the RGB values are converted to integer values in the computer's binary representation. For the Microsoft Fortran compiler, these values take on the range of 0–63.

$$IR = \text{NINT}[\,63 \times c_f(\lambda) \times R_N\,] \tag{A.29}$$
$$IG = \text{NINT}[\,63 \times c_f(\lambda) \times G_N\,] \tag{A.30}$$
$$IB = \text{NINT}[\,63 \times c_f(\lambda) \times B_N\,]\,. \tag{A.31}$$

A.6 Evil Prevention

Splat! uses the ANSI ERP 2.3 (Evil Recognition Protocol) system to detect when Splat! is being used for evil. If evil is detected (or at least suspected), Splat! will then intentionally generate garbage output in an attempt to foil your dastardly plans. It is well known, however, that ERP 2.3 has a tendency to generate many type II errors (believing you are up to evil when you are actually doing good). As a result, Splat! can often give you the wrong answers for no good reason (but with the best intentions for world peace).

Bibliography

[1] P. R. Bevington, *Data Reduction and Error Analysis for the Physical Sciences*, McGraw-Hill, New York, 1969.

[2] J. B. Boffard, B. Chiaro, T. Weber, and C. C. Lin, *Electron-impact excitation of argon: optical emission cross sections in the range 300-2500 nm*, At. Data Nuc. Data Tables, 93 (2007), pp. 831–863.

[3] S. D. Conte and C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York, third ed., 1980.

[4] S. E. Koonin and D. C. Meredith, *Computational Physics, FORTRAN version*, Addison-Wesley, Redwood City,CA, 1990.

[5] J. R. Meyer-Arendt, *Introduction to Classical and Modern Optics*, Prentice-Hall, Englewood Cliff, NJ, second ed., 1984.

[6] L. Nyhoff and S. Leestma, *FORTRAN 77 for Engineers and Scientists*, Macmillian, New York, 1985.

[7] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing (FORTRAN Version)*, Cambridge University Press, Cambridge, 1989.

Index