SCIENCE & ENGINEERING RESEARCH COUNCIL PHOT_FSAN/13.0
RUTHERFORD APPLETON LABORATORY
ISOPHOT Project
ISOPHOT First Science Analysis Note 13.0

S.Guest
13th July 1993

Deglitching of ISOPHOT Data

1 Introduction

ISOPHOT data contains a number of ramps which can be considered 'bad', whether due to particle hits or other anomalies. In this document I refer to all these ramps as *glitches*; note in particular that I make no distinction between *glitches* and *spikes*. The purpose of deglitching is to flag these ramps so they are not used in subsequent processing of the data. No attempt is made to deglitch individual readouts from ramps; only affected ramps from chopper plateaux.

The algorithm used by DERIVE_SPD prior to release 1.3 was simply to exclude ramps with error more than $n\sigma$ from the mean error for the plateau, possibly adequate for chopped observations, but clearly not so for a staring observation, when the signal can drift significantly with time. Furthermore, n was set to be 4.0, far too high since if the errors are Normally distributed with (μ, σ) then

$$\begin{aligned} P(|e-\mu| > 4\sigma) &= 6.34 \times 10^{-5} \\ P(|e-\mu| > 3\sigma) &= 0.0027 \\ P(|e-\mu| > 2\sigma) &= 0.0455 \end{aligned}$$

and of course in the case of a glitch the threshold is distorted since $\frac{\Sigma e}{N} > \sigma$. Neither is it satisfactory to rely on the rms error to detect a disturbed ramp, as there are cases, perhaps if a hit occurs before the first readout, or at the beginning of a ramp with a large number of readouts, where the error is not significantly greater than the average.

A satisfactory algorithm should satisfy the following conditions:

- Removes 'bad' ramps.
- Doesn't remove 'good' ramps.
- Doesn't refine the data to the extent that by removing random error effects, the error of the distribution is underestimated (furthermore, removing ramps increases the signal-to-noise ratio).
- Works on a wide range of 'differently-shaped' datasets.
- Is sufficiently robust to be used with confidence in automated pipeline processing.

- Be tunable if necessary.
- Executes in a 'reasonable' amount of time.

The graphs in this document contain the following symbols:

+ Accepted ramp

∘ Deglitched ramp

□ Saturated ramp

Where a curve has been fitted it is superimposed on the plot. In all cases the X-axis represents time (seconds) and the Y-axis ramp slope (V/s).

2 Three Algorithms

The development of these algorithms was motivated by the problems presented by drifts in staring mode observations; the situation in chopped modes is much simpler and is considered later. A number of different algorithms were experimented with, using four particularly troublesome staring observations with the P1 detector as test datasets. In all cases the first ramp is discarded to start with as it is almost always disturbed. Three algorithms proved reasonably successful on these datasets:

- Polynomial fitting
- Cubic spline fitting
- 'Rolling filter' or 'Local distribution' technique

2.1 Polynomial fitting

This technique involves the following stages:

- Fit a polynomial f of degree d to the (weighted) data. This is done with the NAG routines E02ADF and E02AEF (see [3]).
- Calculate an (unweighted) rms error from the fitted curve, *ie*

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(y_i - f(x_i))^2}{N-1}}$$

- Flag points y_i such that $|y_i - f(x_i)| > n\sigma$.

The plots using this technique (figures 1, 2, 3, 4) also show a polynomial of the same degree fitted to the *deglitched* data. In theory the process could be iterated using this new fitted curve. For an automatic processing system it is necessary to identify values for d and n that work reasonably well for most if not all datasets. By inspection it is clear that d must be odd. If d is too small the fit is too crude; if d is too large the fit develops significant oscillations as t increases. Neither are d and n independent. Experiment has shown the most effective values to be $d = 13$, $n = 2$. The plots were produced using these values.

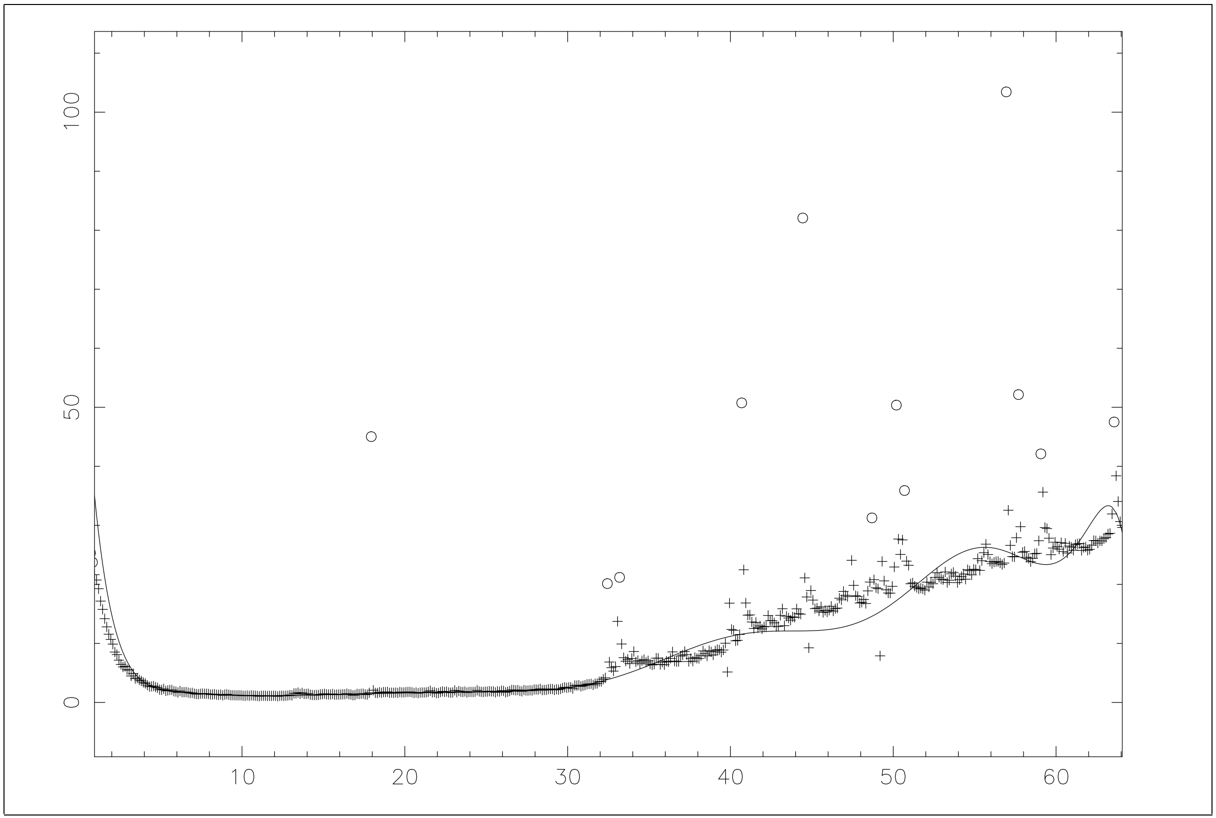

Figure 1: Polynomial fitting, dataset 1

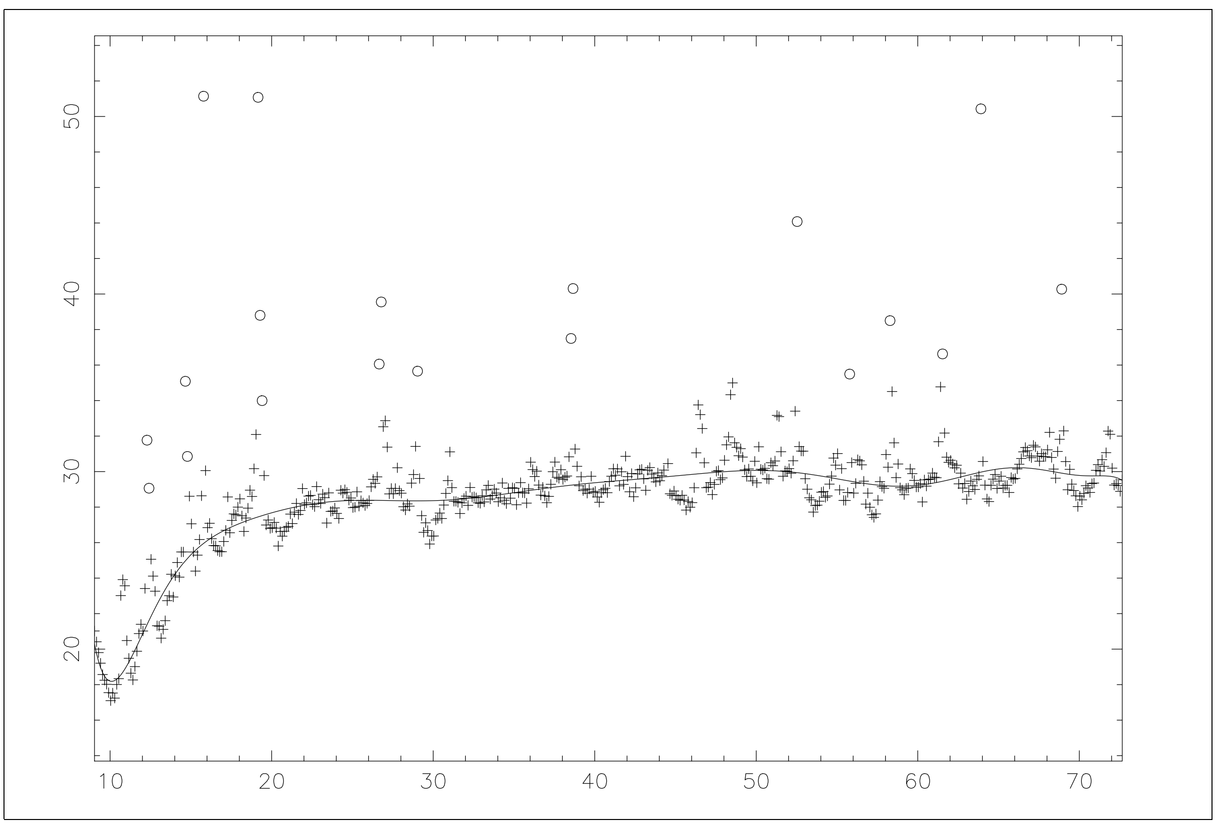

Figure 2: Polynomial fitting, dataset 2

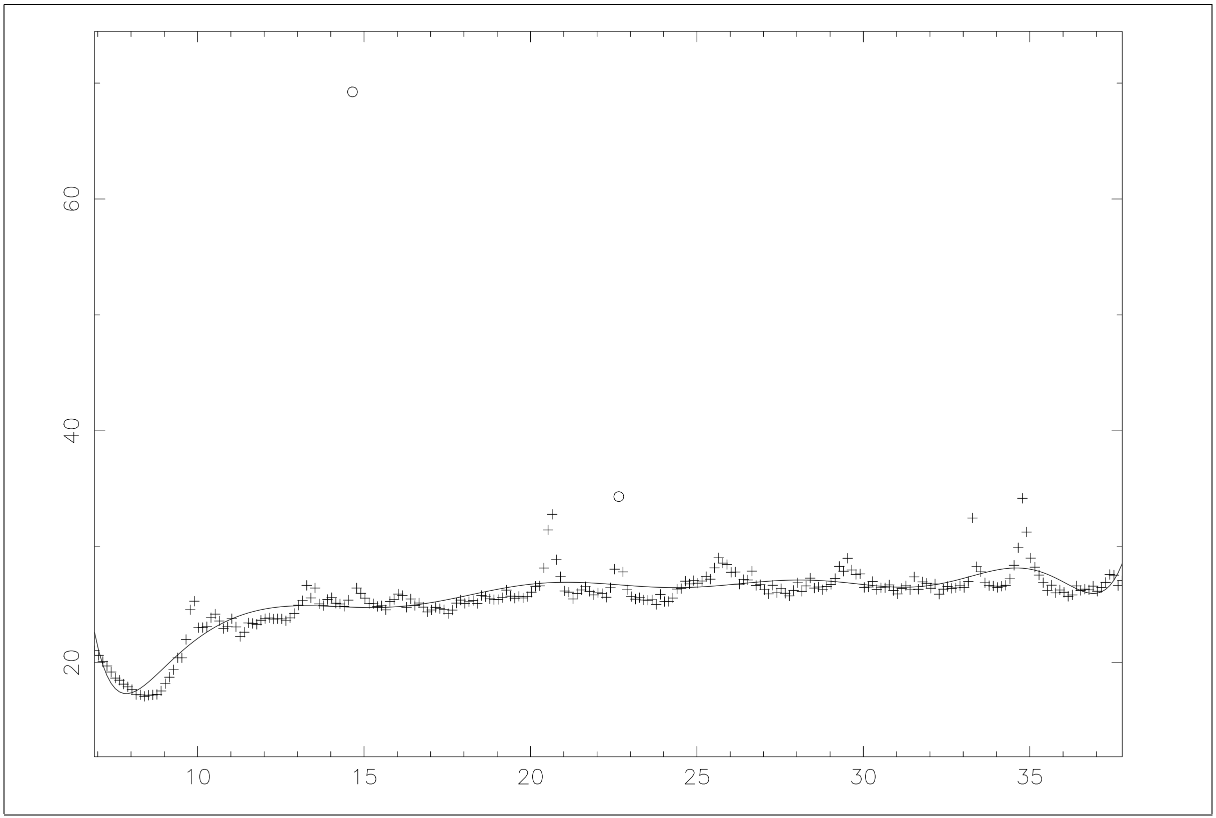

Figure 3: Polynomial fitting, dataset 3

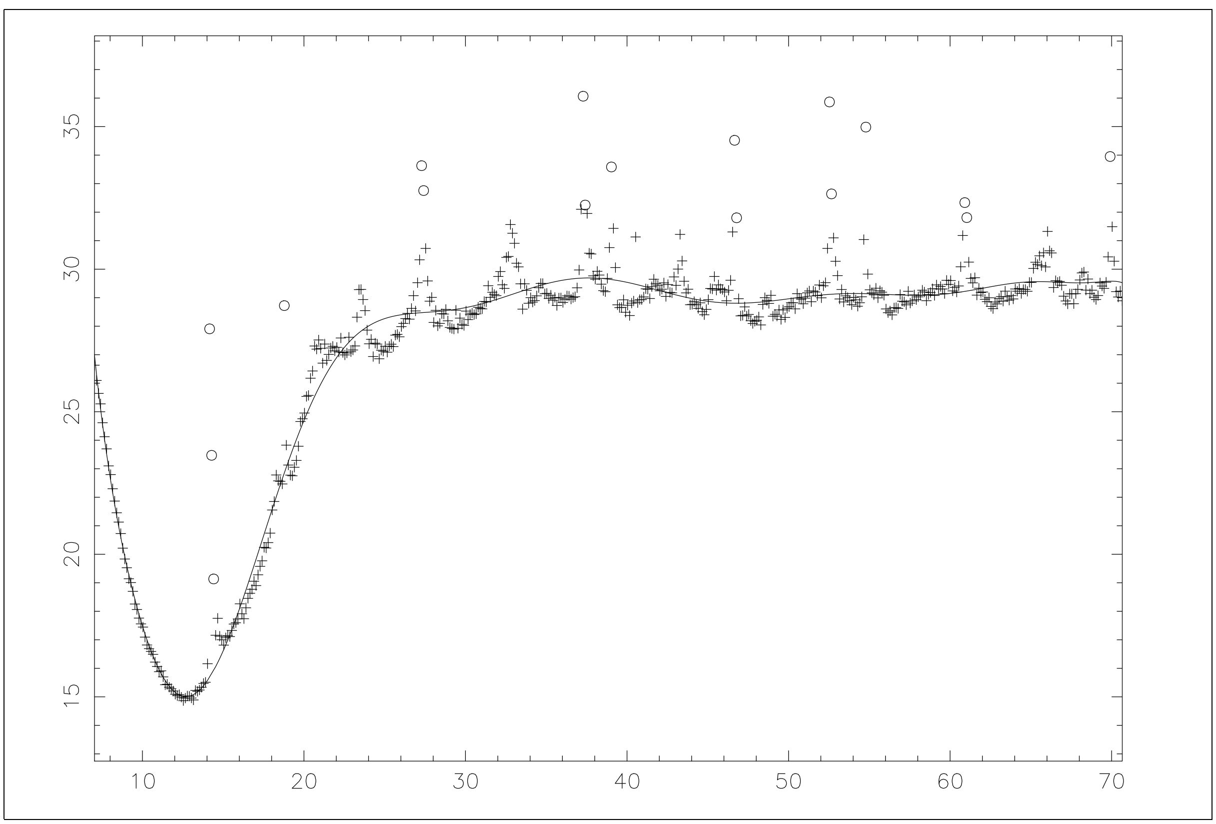

Figure 4: Polynomial fitting, dataset 4

2.2 Cubic Spline fitting

This method is identical to polynomial fitting, except that it uses cubic spline curves calculated using the NAG routines E02BEF and E02BBF (again, see [3]). Instead of degree d, the controlling parameter is a 'smoothness factor' s. Experiment shows $s = 1000$ to be effective. If the process was iterated it would be necessary to decrease s to allow a closer fit. However, fitting too close will result in nothing being rejected; $s = 0$ even results in an interpolating spline! The plots in figures 5, 6, 7 and 8 show a curve fitted to the *deglitched* data with $s = 500$ (the rejection criteria were $s = 1000, n = 2$).

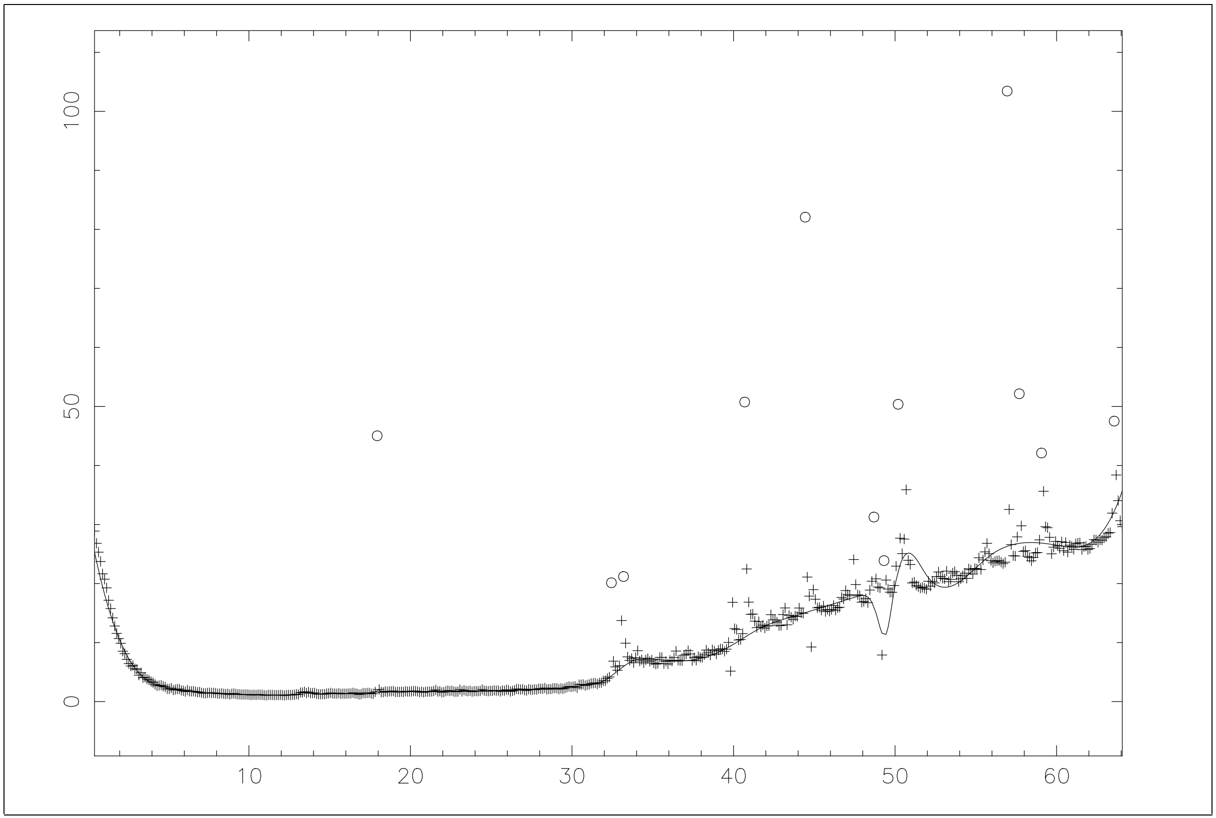

Figure 5: Spline fitting, dataset 1

2.3 Rolling filter

This is a 'local distribution' technique used to filter 'bad' ramps from the data. To understand the main principle, imagine a box sliding along the X (time) axis, calculating local distributions as it goes. The underlying assumption is that distributions are roughly Normal on a sufficiently local scale. The algorithm is as follows:

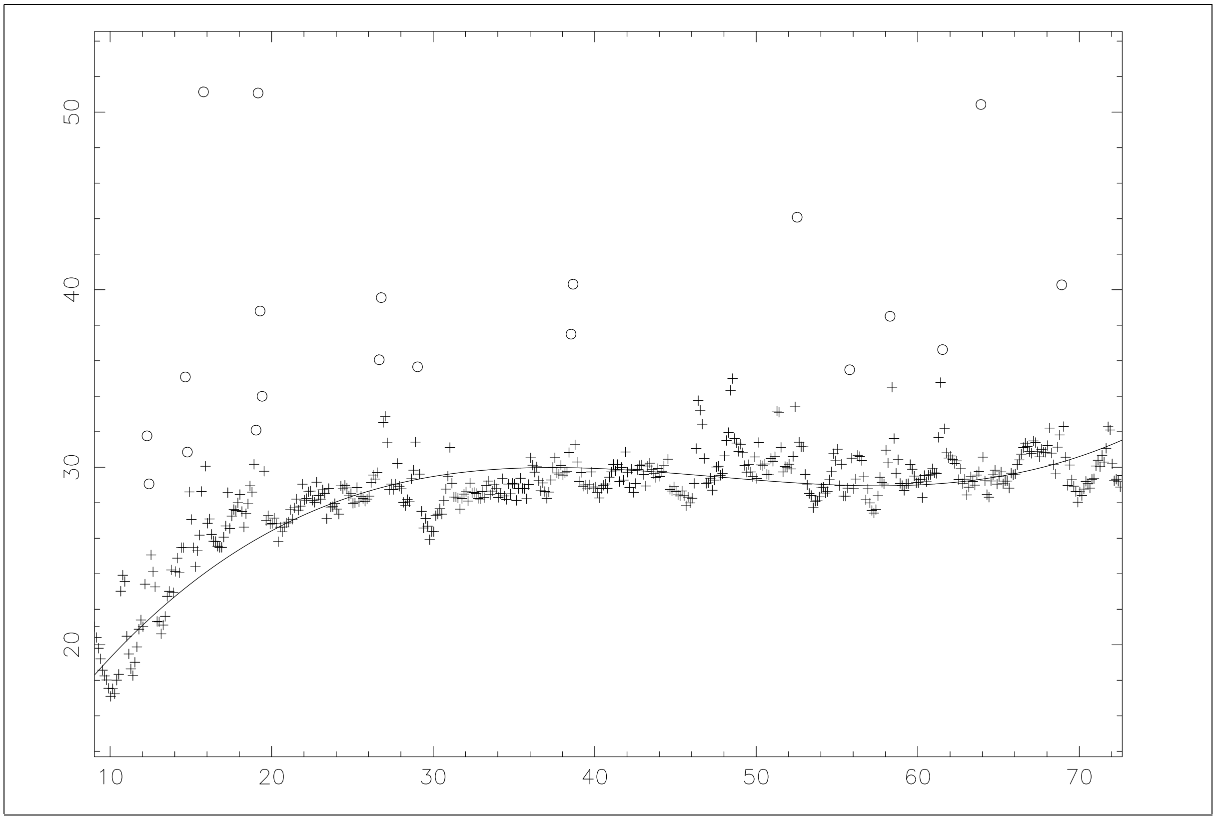

Figure 6: Spline fitting, dataset 2

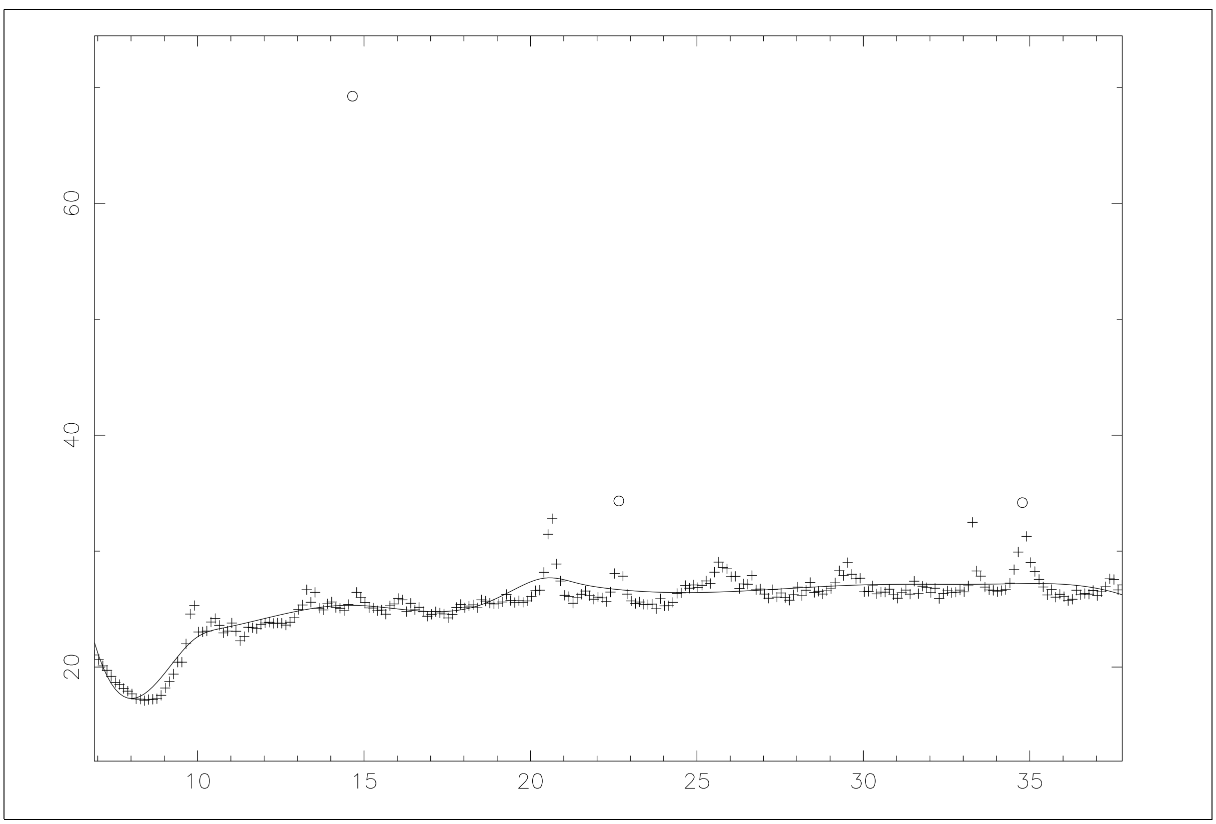

Figure 7: Spline fitting, dataset 3

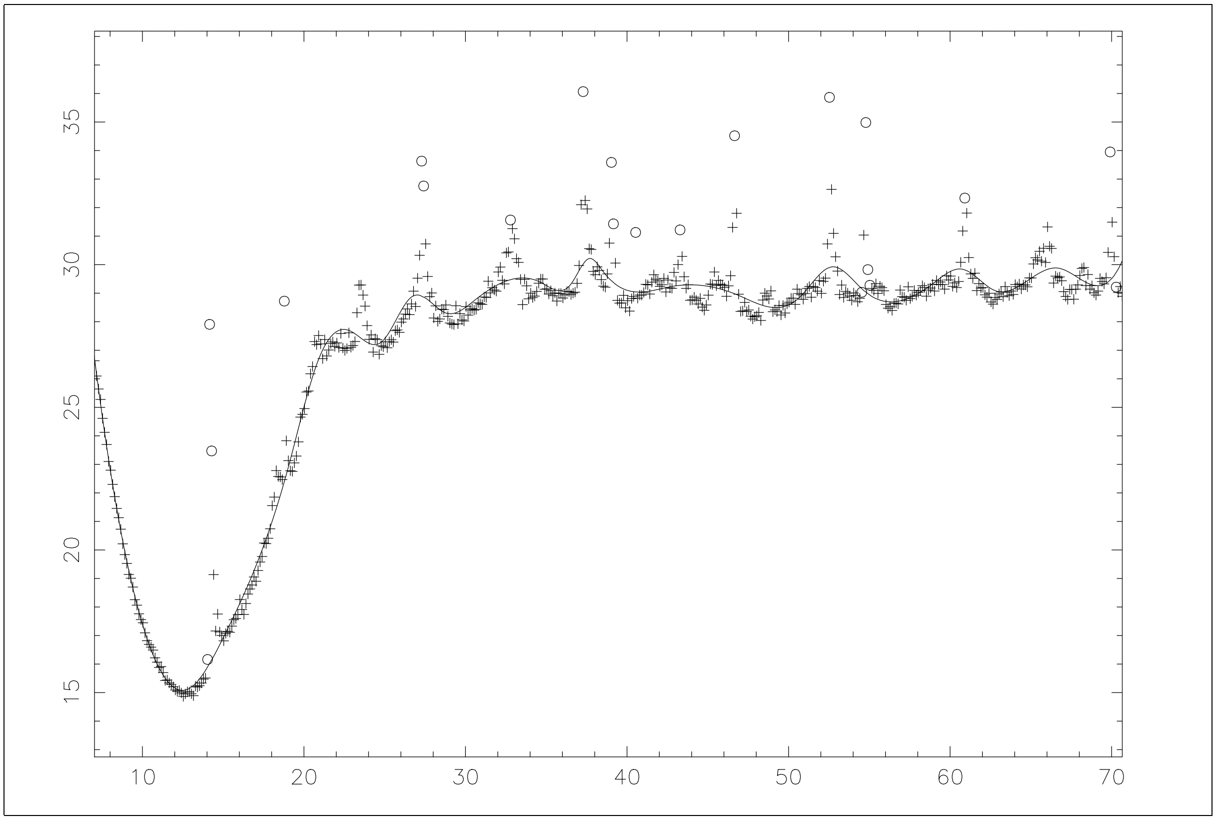

Figure 8: Spline fitting, dataset 4

iterate i times (give up if nothing rejected)
    use only points still regarded as 'good'
    loop until end-data reached
        calculate local median & rms error of $nloc$ points
        identify points $> n\sigma$ from median (†)
        select next local dataset, moving $nstep$ points
    end loop
    flag points with $> nusebad$ failures at test (†)
        (where $nusebad = 1 \quad N \leq nloc$
                $nusebad = nbad \quad N > nloc$
end iterate

The median is used instead of the mean because of its much lower sensitivity to extremely bad points. It can be seen that this algorithm has the following control parameters:

$nloc$	The number of points used to calculate a local distribution.
$nstep$	The number of points to move the 'box' for the local distribution each time.
n	Rejection factor: number of standard deviations from local median.
$nbad$	Number of times point has to be flagged as 'bad' before it is rejected.
i	Number of iterations of the algorithm.

Experiment has shown the values $nloc = 20, nstep = 1, n = 3, nbad = 2, i = 2$ to be effective; these were used the generate the plots in figures 9, 10, 11, 12.

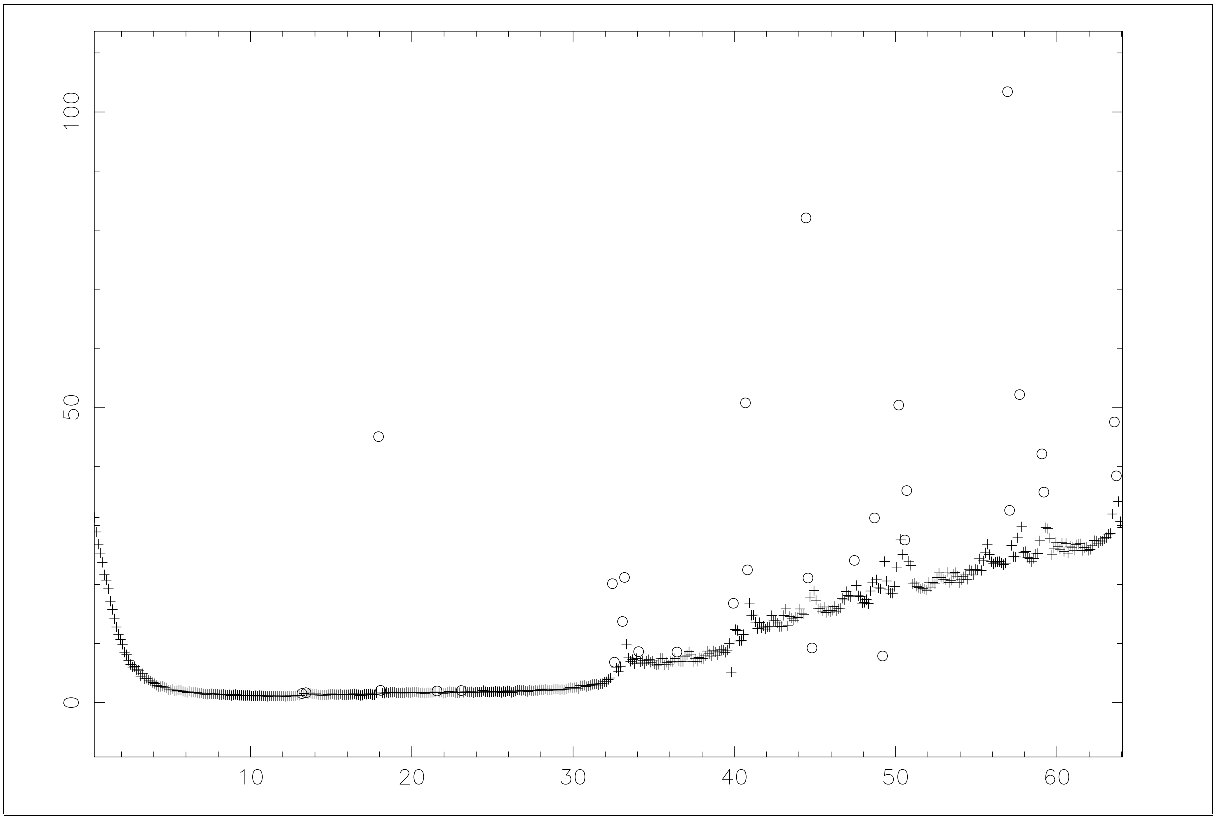

Figure 9: Rolling filter, dataset 1

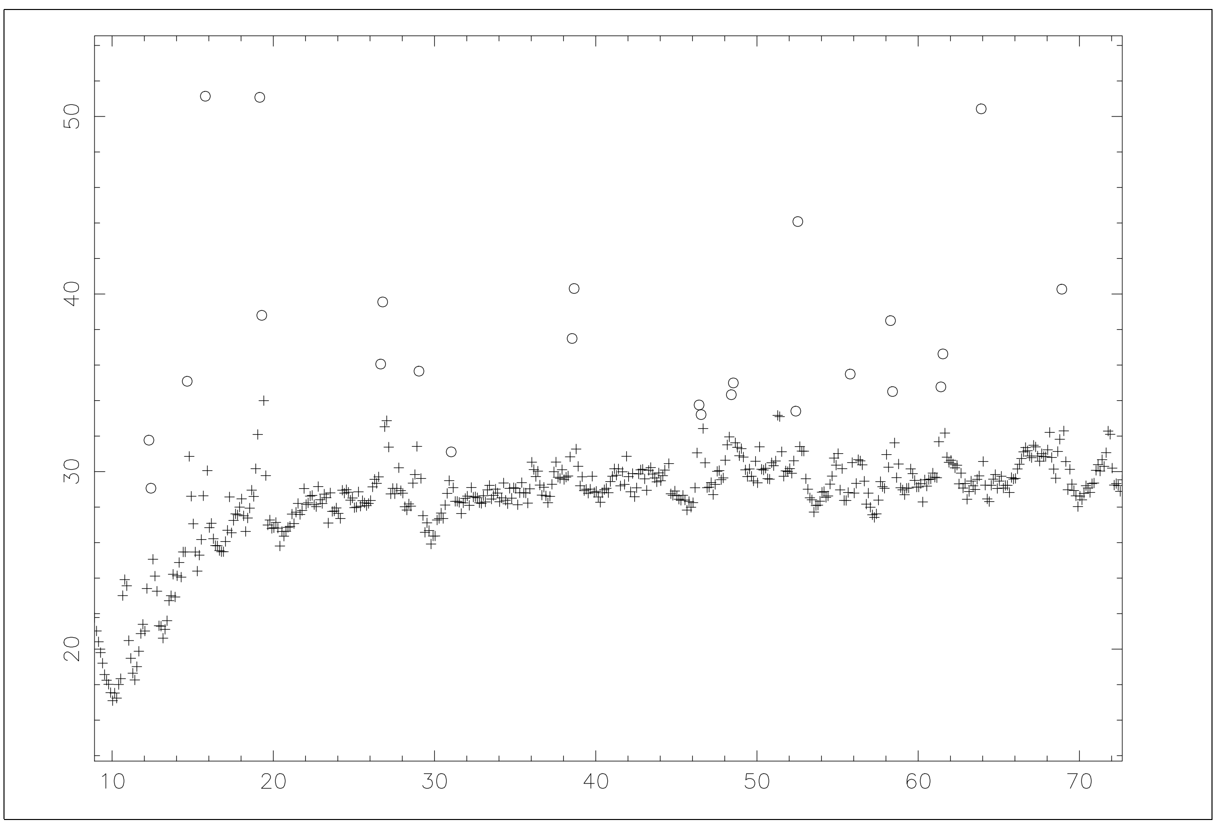

Figure 10: Rolling filter, dataset 2

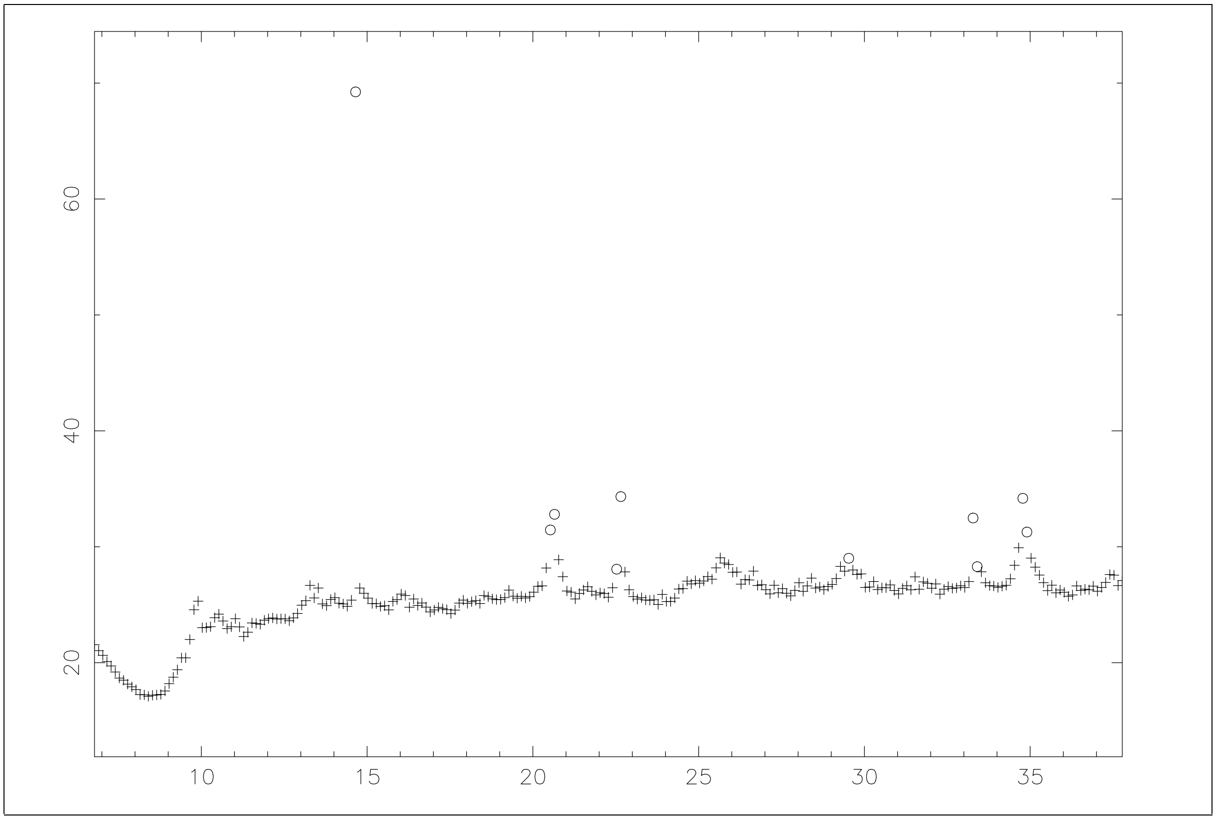

Figure 11: Rolling filter, dataset 3

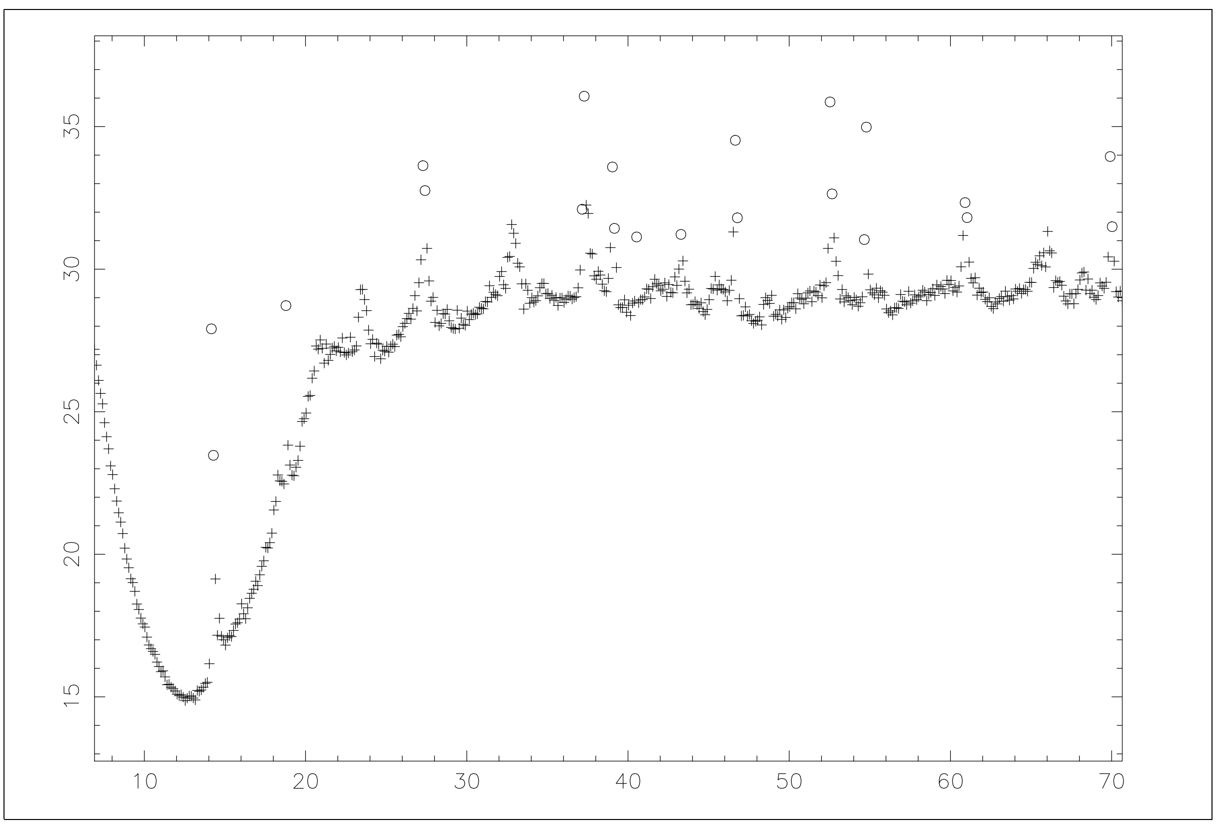

Figure 12: Rolling filter, dataset 4

3 Evaluation

So, having three reasonably effective algorithms, which one should be implemented in DERIVE_SPD? The decision is based on the following factors:

- How effective is the algorithm as a deglitcher? In other words, how well does it perform its primary purpose? Clearly, this is the most important factor.
- How fast does it execute? I really don't want to overstress the importance of this, but an unacceptable overhead may, say, add 50% to the execution speed of DERIVE_SPD.
- How well does it handle chopped modes? If a different algorithm is required it will add to the complexity of the code.

I will consider each of these points in turn, and then calculate some benchmark figure for each algorithm.

3.1 Deglitching

To consider each method in turn...

3.1.1 Polynomial fit

This generally produces a good fit to the 'hook' effect that can be seen in figures 2, 3 and 4, but has more difficulty in figure 1, where the necessity of modelling the very flat section causes problems at either end. Obviously there is a limit to how many maxima and minima a 13^{th} degree polynomial can have, making it insensitive to the kind of peaks seen in figures 3 and 4.

3.1.2 Spline fit

The closeness of a spline fit seems to depend on how good a simple approximation it can manage (compare figures 5 and 8 with 6 and 7). In the case of a good simple fit it can be insensitive to peaks (figure 7), and in a tighter fit over-sensitive to outlying points (figure 5). An advantage of a spline fit is that it can in theory model any shape of curve whatsoever.

3.1.3 Rolling filter

This method deals better with peaks (figure 11), but still not perfectly (figure 12). It is the only method to remove points from under the curve in dataset 1 (figure 9), but it seems to be a little over-zealous in some of the points removed, though this may be an illusion — what it does show is that this algorithm is more sensitive to local anomalies than the others. Conversely, it will be less likely to reject points from a badly behaved local area (see figure 12).

3.2 Execution speed

The following statistics were acquired using the VMS system routines LIB$INIT_TIMER and LIB$SHOW_TIMER:

	Polynomial fit			Spline fit			Rolling filter		
	CPU	BIO	PFs	CPU	BIO	PFs	CPU	BIO	PFs
Dataset 1	0.61	20	104	19.51	7	144	0.70	0	37
Dataset 2	0.59	20	106	0.46	7	101	0.71	0	37
Dataset 3	0.25	20	76	0.88	7	79	0.31	0	27
Dataset 4	0.58	20	104	5.83	7	119	0.68	0	36

'CPU' is used CPU time (seconds), 'BIO' is buffered I/O count and 'PFs' is page faults. It can be seen that spline fitting can have a relatively large CPU overhead in the more extreme datasets, presumably because more knots are required (see [3] for more details). Speeds otherwise are perfectly acceptable.

3.3 Chopped mode considerations

Attempting to fit polynomial or spline curves to plateaux in chopped mode is surely wasted effort. While it is a simple matter to reject points more than $n\sigma$ from a median value, it adds complexity, and should n be the same value as in staring mode? — not necessarily. Here the rolling filter method has the advantage as it works perfectly well on individual chopper plateaux (in most cases $N \leq nloc$ and the calculation is simple). Figure 13 shows a plot of rectangular chopped mode (note that the X-axis here is in clock ticks).

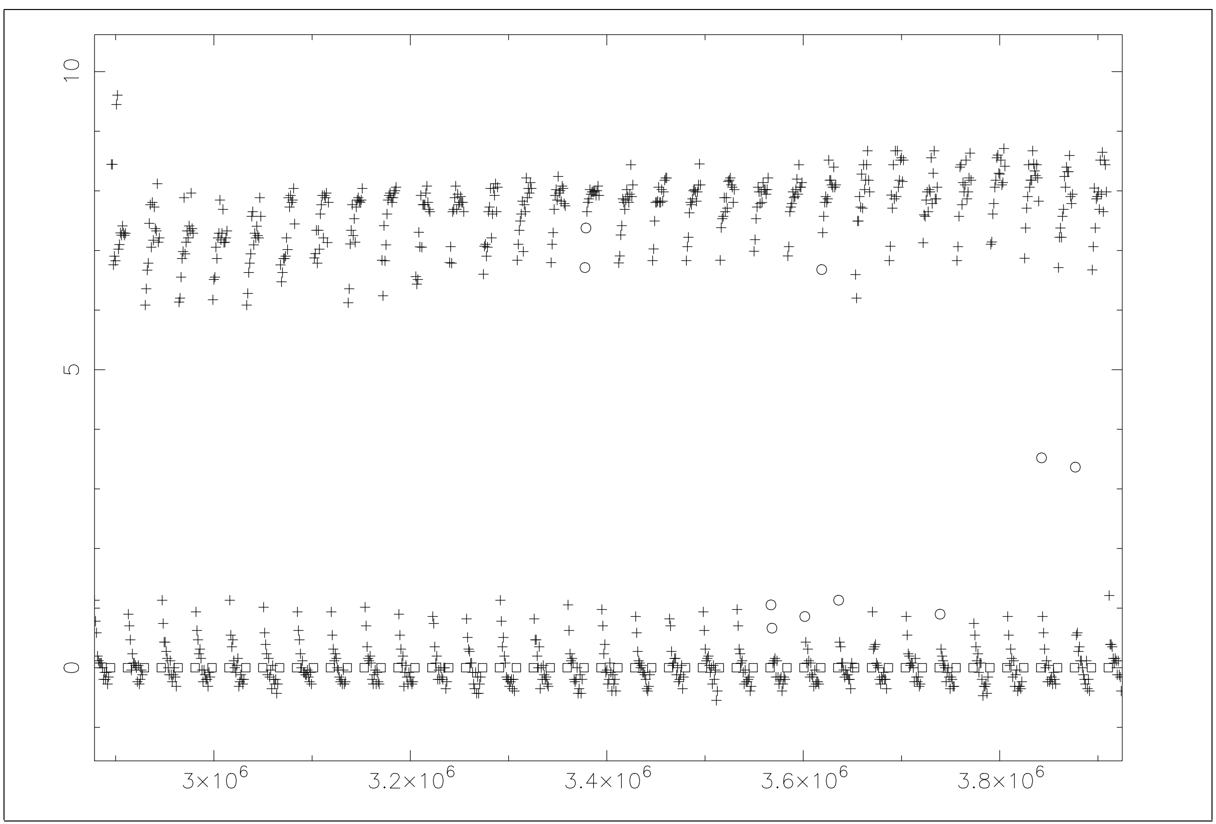

Figure 13: Rolling filter, rectangular chopped mode

3.4 Benchmark

Below I give a score out of 10 for each algorithm for execution speed, chopped mode processing, and deglitching on each dataset (possibly being rather overcritical on the latter scores). These values are then combined with equal weights to give an average score.

	Polynomial fit	Spline fit	Rolling filter
Dataset 1	6	5	7
Dataset 2	8	8	8
Dataset 3	5	6	7
Dataset 4	6	5	5
Execution	7	2	7
Chopped	4	4	10
Average	6.00	5.00	7.33

4 Conclusions

Essentially these conclusions are the result of a large amount of trial and error, working with existing datasets.

- A number of algorithms are possible that can provide reasonably satisfactory deglitching.
- I think it is safe to say that no algorithm can be expected to perform perfectly on all possible data without human inspection and tuning.
- The most suitable algorithm for use in DERIVE_SPD is the rolling filter, which has a further advantage of not requiring NAG. This has now been implemented (see [2]). [1]

[1] The control parameters may be changed via the CAL-G file PCONTROL, and have FITS keywords NLOCAL, NSTEP, NSIGMA, NBAD and NITERATE.

References

[1] Baseline Requirements Document for Off-line Scientific Data Processing, ISO-SSD-9010, issue 1.0.

[2] PHT DERIVE_SPD Detailed Design Document Version 1.3, P. J. Richards, S. Guest, and S. R. Williams, RAL, 2^{nd} July 1993.

[3] NAG FORTRAN Library Manual, Mark 14, Numerical Algorithms Group 1990.

[4] Errors of Observation and their Treatment, J. Topping, Unwin 1955.