

**Studies in Astronomical Time Series Analysis. VI.
Optimum Partition of the Interval:
Bayesian Blocks, Histograms and Triggers**

Jeffrey D. Scargle

Space Science Division, NASA Ames Research Center

Jay Norris

Laboratory for High Energy Astrophysics
Code 661, NASA Goddard Space Flight Center

Brad Jackson, Timothy M. Hsu,
Alina Alt, Sundararajan Arabhi, David Barnes, Peter Gioumousis, Elyus
Gwin, Paungkaew Sangtrakulcharoen, Linda Tam, and Tun Tao Tsai
San José State University, Department of Mathematics and Computer Science,
The Center for Applied Mathematics and Computer Science

Draft of March 20, 2003. **Warning:** this document is under construction. Notation in various sections may not yet be consistent.

Abstract

The problem of detecting statistically significant structure in time series data can be solved by obtaining the partition of the observation interval with the maximum posterior probability of a nonparametric model of the data. We present a surprisingly simple algorithm for obtaining the optimal partition for any data mode, including points, binned points, and measurements at arbitrary times with a known error distribution. The case where the model is a piecewise constant Poisson process is demonstrated in two modes: a real-time or *trigger* mode, and a retrospective mode. The same algorithm also yields histograms, denoised signal shapes, spectral estimates, *etc.*, and goes some distance toward the ultimate goal of a time-domain analysis procedure that optimally represents the underlying signal, suppressing observational noise and the effects of sampling.

Contents

1	Introduction: Bayesian Blocks	3
2	Optimum Partition of an Interval	6
2.1	Data Cells	6
2.2	Blocks of Cells	8
2.3	Partitions	9
2.4	Fitness of a Partition	10
2.5	Changepoints	11
2.6	A Lemma on Subpartitions	11
2.7	The Algorithm	12
3	Posteriors for Sequential Data	14
3.1	Point Data from Discrete Events	15
3.2	Binned Data	17
3.3	Measurements: Point and Distributed	20
3.4	The Posterior	21
3.5	Gaps and Mixed Data Modes	22
3.6	Prior for Number of Blocks	23
4	Examples	25
4.1	TTE Time Series	25
4.2	Binned Time Series	26
4.3	Real Time Analysis	27
4.3.1	Triggers	27
4.3.2	Running Block Analysis	27
4.4	Histograms	27

5	Appendix A: MatLab Code	28
5.1	Main Program	28
5.2	Construct Data Cells	30
6	Bibliography	33

1 Introduction: Bayesian Blocks

We present an improved version of *Bayesian Blocks* [Scargle 1998], hereafter Paper V, a method of detecting and characterizing variability – both random and deterministic [Scargle 1981] – in time series data corrupted by observational errors. The approach underlying this algorithm attempts to achieve the ideals for exploratory analysis listed below. An underlying idea is that this approach and the algorithm implementing it will be useful in the context of automated science data analysis, and in turn be useful in astronomical data base analyses such as those implemented by the Virtual Observatory.

Here is the general philosophy of the analysis:

- make few prior assumptions about the signal:
 - minimal smoothness assumptions
 - no shape assumptions
- handle large dynamic ranges in
 - amplitude
 - scale
- automatic
 - for scientific data mining applications
 - for objectivity
- eliminate noise
- simple to use, easy to compute
- impose no *a priori* limits on:

- scales
- resolution
- detect local, rather than global, structure
- conserve information content
- not greedy¹
- automatic penalty for model complexity
- run time $\sim N^2$ or less
- handle gaps and exposure variations
- allow two modes:
 - retrospective (model all data)
 - real-time - *triggers on 1st significant excursion*
- output suitable for further analysis
- exactly reversible
- compress data volume
- allow incorporation of auxiliary, extrinsic data, such as variable exposure, energy dependences, *etc.*

Some of these items were part of the original goal; many were unexpected benefits of the approach adopted. These issues will be discussed individually in more detail below, when we are describing how the new algorithm achieves these properties.

¹A *greedy* algorithm is an iteration in which an optimum operation is carried out at each step, but because such a local procedure may make mistakes that can't be corrected later, a global optimum is not guaranteed.

This list suggests the use of the most generic possible nonparametric data model, and has motivated our development of data segmentation and Bayesian changepoint methods [Ò Ruanaidh and Fitzgerald 1996]. It is remarkable that a very simple idea – *fitting of piecewise constant models to the data* – achieves essentially all of the above desiderata. This approach yields a step-function, or segmented, representation of the signal in which the range of the independent variable (*e.g.* time) is automatically divided into unequal subintervals, in each of which the dependent variable (*e.g.* intensity) is modeled as constant. Note that density estimation, or the formation of histograms can be viewed as essentially the same problem – as can cluster detection and, less directly, classification.

This representation is in the spirit of a nonparametric approximation, and not meant to imply that we believe the signal is really discontinuous. The crude, blocky appearance of the discontinuous model may be a liability in the context of visualization, but for our interests in deriving physically meaningful quantities we have not found it so. Blocky models are useful in broad signal processing contexts [Donoho 1994], and have several motivations. Their simplicity allows exact treatment of the likelihood. We can marginalize the rate parameters exactly, giving convenient analytic formulas for the posterior. And we regard the estimated model itself as less important than quantities derived from it. For example, while smoothed plots of pulses within gamma-ray bursts make pretty pictures, one is really interested in pulse locations, lags, amplitudes, widths, rise and decay times, *etc.* These quantities can be accurately determined directly from the locations, heights and widths of the blocks.

One could consider piecewise linear models (*cf* ref Mannila et al).

Our view is that such models may have a better visual appearance, but all in all the improved flexibility in fitting is largely offset by the added complexity of the model and its interpretation. Note further that if continuity is imposed at the changepoints, a piecewise linear model has essentially the same number of parameters, or degrees of freedom, as does the simpler piecewise constant model.

The following section describes the new algorithm, and the rest of the paper describes block analysis of time series and other data, with examples. Below §2 describes how sequential data are presented for input to the algorithm, §3 deals with the fitness, or cost, function used in the analysis, and §4 gives examples of histograms, time series analysis, and triggers.

2 Optimum Partition of an Interval

Our algorithm works on any sequential data. We introduce it in a somewhat abstract setting because it can be used for other partitioning problems beyond time series analysis. In a special case it implements Bayesian blocks or other 1D segmentation ideas with any model fitness function that satisfies a simple additivity condition. It improves on previous approximate Bayesian block algorithms by achieving a rigorous solution of the multiple changepoint problem, and is guaranteed to find the global maximum, not just a local one. This approach may be useful in higher dimensional data spaces, but there are significant difficulties even in 2D that have not yet been solved.

2.1 Data Cells

Consider N data elements

$$\mathbf{x}_n, \quad n = 1, 2, \dots, N. \quad (1)$$

each consisting of a set of numerical quantities or other attributes defining the observed data. The meaning of the array \mathbf{x}_n is left vague because almost any of a wide variety of data types can be treated with the current formalism. Simple examples are: points, counts of points in bins, and measurements – correspondingly, the array \mathbf{x} would contain point coordinates; counts, bin sizes and locations; and measured values and their uncertainties, respectively. The only requirement is that the data be ordered (*i.e.*, *sequential*), meaning that each \mathbf{x}_n is associated with a time t_n , such that the latter are ordered and contained in some time interval I :

$$\min(I) \leq t_1 < t_2 < \dots < t_N \leq \max(I) \quad . \quad (2)$$

In general t_n specifies the time of measurement, be it a point or an interval. For point data (also called event data), t_n is just the time of event n . Although times are often represented as real numbers, the finite accuracy of measurement means that one is really specifying an integer multiple of some small unit of time (typically on the order of milliseconds to microseconds in high energy astrophysics). For cases such as binned counts or measurements averaged over finite time intervals, the time interval must be specified, either explicitly (as in an array giving the lengths of a series of unequal time bins) or implicitly (*e.g.* through specification of bin size and time of the first bin).

It is convenient to represent sequential data with a data structure consisting of a set of N *data cells*

$$C_n \equiv \{\mathbf{x}_n, t_n\} \quad , \quad (3)$$

derived from the raw data. They form an ordered sequence with respect to the independent variable t , can be grouped into blocks (§2.2) forming partitions of I (§2.3), and contain whatever data quantities are necessary to evaluate the fitness (§2.4) of an arbitrary partition. In some cases two or more data elements are combined into a single cell (see *e.g.* the discussion of duplicate time tags in §3.1), but for the most part data cells correspond one-to-one with data elements. In some cases (*e.g.* time-tagged event data) t_n is contained in \mathbf{x}_n and need not be separately specified. Figure 1 is a cartoon of typical data cells.

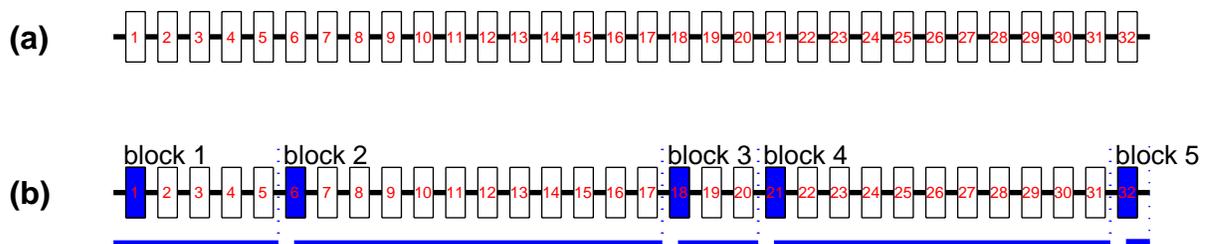


Figure 1: Pictorial representation of data cells and the blocks made from them. The horizontal axis represents the independent variable (often, but not necessarily time), with respect to which the data are ordered. The sequential order depicted in Panel (a) is the only essential requirement for data to be analyzable with our block algorithm. Panel (b) exemplifies the partition of the set of data cells into blocks. The shaded cells are changepoints marking the beginnings of the blocks.

2.2 Blocks of Cells

A block is a set of adjacent cells. Panel (b) of Figure 1 shows a sequence of 32 data cells divided into five blocks. The following notation for

blocks is useful:

$$\mathbf{B}(n, m) \equiv \{C_n, C_{n+1}, \dots, C_m\} , \quad (4)$$

that is $m - n + 1$ cells in sequence. The case $m = n$ represents a block consisting of just one cell, as in the last block of the partition in Figure 1(b). An empty block has no meaning.

Why do we chunk the data into blocks? The cells in a block are treated as a whole, in that they are described by a single statistical model. Typically the same parametric model applies to all blocks, the model parameters are constant within each block, and the values of these parameters undergo jumps at the changepoints (§2.5) marking the edges of the blocks. And finally, the fitness of a block is of elementary importance, because the fitness of a partition (§2.4) is the sum of the fitnesses of the blocks comprising it.

2.3 Partitions

A *partition* of the interval I is simply a set of nonoverlapping blocks that together add up to the whole interval.² A partition can be defined by specifying N_{blocks} , the number of blocks (the elements of the partition), and the block edges, n_k :

$$\mathcal{P}(I) \equiv \{N_{blocks}, n_k, k = 1, 2, 3, \dots, N_{blocks}\} . \quad (5)$$

There are one fewer changepoints than blocks, since by convention the first block begins at the first data cell – $n_1 \equiv 1$ is implicit – and the last block terminates with the last data cell. As described in §2.4 we will seek the partition that maximizes a given function over all possible partitions. How big is this search space if there are N cells? Establish

²Formally a partition of I is a set of blocks satisfying $I = \bigcup_k B_k$ and $B_j \cap B_k = \emptyset$ (the null set), for $j \neq k$.

a 1-1 mapping between partitions and binary numbers of length N , by setting the k -th digit to 1 if cell k is a changepoint, 0 otherwise. Remembering that the first cell is always a changepoint, the number of partitions is then

$$N_{\text{partitions}} = 2^{N-1} \quad (6)$$

Except for short time series this number is too large for an exhaustive search, but our algorithm nevertheless finds the optimum over this space in a time that scales as only N^2 .

2.4 Fitness of a Partition

Since our goal is to model data, we maximize³ a quantity measuring the fitness of models in a specified class. We take as this model class all partitions of the interval, with a given statistical model for each block of the partition. If the observational errors at different times are independent, as is often the case, fitness is additive over blocks:

$$F[\mathcal{P}(I)] = \sum_{k=1}^{N_{\text{blocks}}} f(B_k) \quad , \quad (7)$$

where $F[\mathcal{P}(I)]$ is the total fitness and $f(B_k)$ is the fitness of block k . Our algorithm depends explicitly on this additivity.

Specific examples and details of fitness functions are given below in §3. What is important here is that we marginalize all parameters of the block models except the times defining the beginning and end of the block (Paper V). Then the total fitness depends on only $\mathcal{P}(I)$. The best model is found by maximizing F over all partitions. As an example, the fitness function we adopt for count data does not depend on the Poisson rate parameters – they can be computed in an almost trivial way, once the changepoints of the optimum partition are determined.

³Alternatively, one can minimize an error measure. Both are called *optimization*.

2.5 Changepoints

We call the time separating two blocks a *changepoint*⁴. In principle a changepoint could be anywhere in the interval, but we restrict them to occur at the times corresponding to the data cells. The reasoning is that moving a changepoint lying between two data cells to a new location between the same cells does not sensibly change the model’s representation of the data. This simplification reduces the search over an infinite space to a finite optimization problem.

In some applications it might be useful to assign a data cell that is a changepoint to be in *both* the subsequent and previous blocks, but here we assign it to only one – with the convention that a changepoint is the first cell in the subsequent block (rather than the last cell of the previous block). Correspondingly, since the smallest partition consists of a single block containing all data cells, the first data cell is always a changepoint. If the last cell is a changepoint, it demarcates a block consisting of that one cell, as in panel (b) of Figure 1, where the five changepoints dividing the data cells into five blocks are shaded.

2.6 A Lemma on Subpartitions

We define a *subpartition* of a given partition $\mathcal{P}(I)$ is a partition (of a subset of I) consisting of a subset of the blocks of $\mathcal{P}(I)$. Although not a necessary condition for the lemma to be true, in all cases of interest here the blocks in the subpartition are contiguous, and thus form a partition of a subinterval of I . Below we will make use of this simple result on subpartitions of optimal partitions:

⁴In statistics, a changepoint in a time series is a point at which the statistical model undergoes an abrupt transition, usually by one or more of its parameters jumping to a new value

Lemma: A subpartition of an optimal partition is an optimal partition of the subset it covers.

Let \mathcal{P}' be the subpartition and I' the subset of I that it covers. If there were a partition of I' , different from and fitter than \mathcal{P}' , then combining it with the blocks of \mathcal{P} not in \mathcal{P}' would, by the block additivity condition, yield a partition of I fitter than \mathcal{P} , contrary to the optimality of \mathcal{P} . ■

Corollary: removing the last block of an optimal partition leaves an optimal partition.

2.7 The Algorithm

We have assembled the definitions and results needed to state our procedure and prove that it finds a global optimum partition. This algorithm is in the spirit of dynamic programming [Hubert, Arabie, and Meulman 2001]. It begins with the first data cell, adding one more at each step until the whole interval has been treated. This feature makes the algorithm suitable for real-time applications (see §4.3).

The proof is by mathematical induction: if a theorem is true for $R = 1$, and one can show that, if it is true for R then it is true for $R+1$, then the theorem holds for all R . At step R the algorithm finds the optimum partition of the interval comprised of data cells $I_R \equiv \{C_1, C_2, \dots, C_R\}$. To analyze all the data, take $R = 1, 2, \dots, N$. The case $R = 1$ is trivial: there is only one cell, and the only partition possible is the optimum one.

Now suppose we have completed step R , having obtained the opti-

mal partition $\mathcal{P}^{opt}[I_R]$, hereafter abbreviated $\mathcal{P}^{opt}(R)$, and are now at step $R + 1$ and wish to find the optimal partition $\mathcal{P}^{opt}(R+1)$. Assume further that we have kept a running record of the fitness of the optimum partition obtained at each previous step (call this array **best**) and the location of the last changepoint in that partition (call this array **last**). It is straightforward to compute

$$M(r) \equiv f[\mathbf{B}(r, R + 1)] \quad (r = 1, 2, \dots, R + 1) \quad (8)$$

that is, the fitness of a putative last block starting at r and extending to the end of the current interval. For example $M(1)$ is the fitness of the whole interval currently in play, namely the cells from 1 through $R + 1$.

Using the block additivity of fitness, Eq. (7), the fitness of the partition of I_{R+1} consisting of the optimum partition $\mathcal{P}^{opt}[I_{r-1}]$ followed by a single block $\mathbf{B}(r, R + 1)$ is:

$$A(r) = M(r) + \begin{cases} 0 & r = 1 \\ \mathbf{best}(r - 1), & r = 2, 3, \dots, R + 1 \end{cases}, \quad (9)$$

Now comes the key reasoning step. While we don't yet know what it is, the new optimum partition $\mathcal{P}^{opt}(R + 1)$ must exist and must have a *last changepoint*, say r^* .⁵ From its definition $A(r^*)$ is the fitness of $\mathcal{P}^{opt}(R + 1)$. In particular, $\mathbf{best}(r^* - 1)$ is the fitness of the optimal subpartition consisting of all but the last block of $\mathcal{P}^{opt}(R + 1)$, and $M(r^*)$ is the fitness of said last block. Further, any partition with its last changepoint at some other $r \neq r^*$ must have fitness not greater than that of $\mathcal{P}^{opt}(R + 1)$, so we have

$$A(r) \leq A(r^*) \quad \text{for } r \neq r^* . \quad (10)$$

⁵Any finite combinatorial optimization problem has at least one solution. Also, all partitions have at least one changepoint.

In other words, the maximum of $A(r)$ occurs at r^* :

$$r^* = \operatorname{argmax}[A(r)] , \quad (11)$$

so finding the fitness and last changepoint of $\mathcal{P}^{opt}(R+1)$ is just a matter of finding the maximum of the array A and the index r at which this maximum occurs.

At the end of the computation, it only remains to find the locations of the optimal changepoints. The needed information is contained in the array $\mathbf{last}(r)$ in which we have stored the index r^* at each step. Using the corollary of the subpartition lemma, it is a simple matter to use the last value in this array to determine the last changepoint in $P^{opt}(N)$, peel off the end section of \mathbf{last} corresponding to this last block, and repeat. That is to say, the values

- (1) $cp_1 = \mathbf{last}(N)$
- (2) $cp_2 = \mathbf{last}(cp_1 - 1)$
- (3) $cp_3 = \mathbf{last}(cp_2 - 1)$
- ...

are the index values giving the locations of the changepoints, in reverse order. The positions of the changepoints are not necessarily fixed until the very last iteration, although in practice it turns out that they become more or less “frozen” once a few succeeding changepoints have been detected.

The MatLab code for the algorithm in Appendix XX indicates how all of these computations are implemented.

3 Posteriors for Sequential Data

After a few general remarks about data, this section discusses the specifics of determining the data cells appropriate to several data types, and the computation of the corresponding posterior probability.

The algorithm works on a wide variety of data types. All that is necessary is that the data be sequential with respect to some independent variable, as outlined in §2.1, and one must know enough about the observational errors to compute the posterior probability for the model of a block of data. The data may be computationally determined, as in a power spectrum derived from time series data, or even the output of a computer simulation.

We consider measurements made to determine the magnitude of some physical quantity, the *dependent variable*, as a function of some other quantity, the *independent variable*. Typically the latter is controlled in the measurement process (*e.g.* time, wavelength, or position) and the dependent variable results from the physical process of interest, (*e.g.* radiant intensity, or the density of some kind of discrete events – see §3.1).

The set of possible values of the independent variable is called the *data space*. For the one dimensional case to which this discussion is confined, the data space is usually an interval, such as the time over which observations have been made.

3.1 Point Data from Discrete Events

Sometimes the physical process, or perhaps the way of recording it, takes the form a sequence of discrete events, each yielding a point in the data space. In practice point coordinates are integer multiples of

some small but finite unit – and are thus discrete, not continuous. The quantity of ultimate interest is the *distribution function* of the points, interpretable as the intensity or probability density of some physical variable – hence the term *density estimation* is sometimes used in such cases. A key example is the case where the events are the detection of individual photons, the corresponding points are the measured detection times, and the quantity of interest is the radiation intensity as a function of time.

For point data, it is natural to associate one cell with each event. However, if the detector allows the separate detection of two (or more) events that are simultaneous to within the accuracy with which times are recorded, such pairs can be assigned to the same cell. As stated earlier (§2.1), a set of data cells must contain everything necessary to compute the fitness (§3) of the block containing the cells. One requirement is the number of events in the block, so the first entry in the cell data structure is the number of events assigned to the cell (most often 1). Since another requirement is length of the block, the second entry is the length of the interval associated with the event.

There is more than one way to define an interval associated with a given event in a sequence. Perhaps the most natural choice is *the set of times closer to that event than to any other*. This definition yields intervals defined by the midpoints between successive events: from preceding midpoint to the succeeding one. This choice is also motivated by the ease of its generalization to higher dimensions (the *Voronoi tessellation* of the data points, [Okabe, Boots, Sugihara and Chiu 2000, Scargle 2001a, Scargle 2001c]), and its representation of the local point density.

Alternatively, one can use the intervals between successive data

points, roughly speaking assigning half of an event to the interval immediately to its left and half to the one immediately to its right. This choice may handle the onset of a steep gradient in the underlying density slightly better, and is also easily generalizable to higher dimension as the *Delaunay triangulation* [Okabe, Boots, Sugihara and Chiu 2000]. We allow a choice of these two modes in the code in the Appendix A.2.

Note that the quantities needed for computation of the fitness of a block are simply the sums of the corresponding quantities over the cells making up the block. This convenient result is a consequence of the piece-wise constant Poisson model. For other models it might not hold, but this would only make the computation of the cost function more complex, and would not affect in any way the partitioning algorithm.

In effect, we are ignoring the actual values of the independent variable, since only interval lengths and cell populations are needed. In principle, the cells in a block need not even be contiguous. For example, the cells near the very beginning and the very end of the interval might belong to a constant background intensity level, and one would therefore like to be able to interpret them as belonging to the same block. An algorithm allowing wraparound would be a natural way to deal with this case. For most purposes it is convenient to impose cell contiguity on the blocks, and our algorithm has this condition built in.

For photon counting data (Paper V) used the posterior probability of the constant-rate Poisson model, with the rate parameter marginalized, to measure model fitness. As is typical for photon counting data modes, this marginal for a block is a function of only N , the number of photon counts in the block, and M , the length of the block:

$$f_{point}(N, M) = \frac{\Gamma(N + 1)\Gamma(M - N + 1)}{\Gamma(M + 2)} \quad (12)$$

This is the quantity to be used in Eq. (8).

3.2 Binned Data

For binned data it makes sense to let the data cells correspond to the bins. This section derives the fitness function for binned data, which has a different algebraic form than for point data. We generalize the result in Paper V in two ways, allowing unequal bins and a variable efficiency factor.

The first entry in the cell data structure is the count of the number of events with coordinates lying in the bins. If the bins are all equal, the constant bin width is assumed, but for unequal bins the size of each bin must be specified, and is the second entry in the cell data structure, namely M_n .

We can also generalize the analysis to include a variable efficiency. In astronomy, this factor is sometimes called *exposure*, since it arises when the time a telescope spends collecting photons varies across the sky; in addition, instrumental sensitivity may vary with time. We assume that whatever the effect is, it can be represented by specifying an efficiency factor, $0 < E_n \leq 1$, for each bin, such that the effective Poisson event rate for the data collected in bin n is

$$\lambda_{\text{eff}} = \lambda E_n , \quad (13)$$

where λ is the true, observed or modeled, event rate.

The likelihood for bin n is

$$L_n = \frac{(\lambda E_n W_n)^{N_n} e^{-\lambda E_n W_n}}{N_n!} \quad (14)$$

where here λ is the true Poisson parameter (events per unit time) specified by the model for the block containing the bin, and W_n is the

width of the bin (in the same units). The likelihood for block k is

$$L_k = \prod_{n=1}^{M_k} L_n = \lambda^{N_k} e^{-\lambda \sum_n E_n W_n} \prod_n [(E_n W_n)]^{N_n} . \quad (15)$$

Here the product and the sum are over all bins in the block, M is the number of bins in the block, and

$$N_k = \sum_{n=1}^{M_k} N_n \quad (16)$$

is the total event count from the data in block k , and we have thrown away the denominator in Eq. (14) – for the usual reason that this factor is the same for all models, and is thus irrelevant for model comparison. As in Paper V, we marginalize over the Poisson rate, but now use the following more general form for the prior distribution of λ :

$$P(\lambda) = P_0 \lambda^{\alpha-1} e^{-\beta\lambda} \quad (17)$$

where

$$P_0 = \frac{\beta^\alpha}{\Gamma(\alpha)} \quad (18)$$

is the normalization constant. Gelman et. al (1995) argue for the appropriateness of this prior for Poisson data, noting that it is conjugate to the Poisson likelihood, and also has the interpretation that this “prior density is, in some sense, equivalent to a total count of $\alpha-1$ in β prior observations.”

Using this prior, the marginalized posterior probability of the Poisson model \mathcal{M}_k for block k , given the data $D_k = \{N_n, n = 1, 2, \dots, M_k\}$, is

$$P(\mathcal{M}_k | D_k) = P_0 \prod_n (E_n W_n)^{N_n} \int_0^\infty \lambda^{N_k + \alpha - 1} e^{-\lambda(\sum_n E_n W_n + \beta)} d\lambda . \quad (19)$$

Carrying out the integral, we have

$$P(\mathcal{M}_k | D_k) = P_0 \frac{\Gamma(N_k + \alpha) \prod_n (E_n W_n)^{N_n}}{(\sum_n E_n W_n + \beta)^{N_k + \alpha}} \quad (20)$$

Note that the prior in equation (28) of Paper V, corresponds to $\alpha = 1$ and $\beta = 1$, and the equation above reduces to equation (29) of that paper for equally spaced bins.

Application of the algorithm described in §2.7 merely requires the straightforward computation of Eq. (20) using the known values of E_n and W_n . Examples will be given in §4 below.

3.3 Measurements: Point and Distributed

The data can also consist of measurements of a quantity at a set of points. For example, the data array could be

$$x = \{x_n, t_n, \sigma_n\} \quad n = 1, 2, \dots, N, \quad (21)$$

where x_n is the value measured at time t_n , and σ_n is some measure of the corresponding observational uncertainty. The measurement is never strictly at a point, but distributed over an interval. If warranted, the variation of the instrumental sensitivity over the interval must also be specified – in terms of a *window function*.

We assume the standard piece-wise constant model of the underlying signal, that is, a set of contiguous blocks:

$$B(x) = \sum_{j=1}^{N_b} B^{(j)}(x) \quad (22)$$

where each block is represented as a *boxcar* function:

$$B^{(k)}(x) = \begin{cases} B_j & \zeta_j \leq x \leq \zeta_{j+1} \\ 0 & \text{otherwise} \end{cases} \quad (23)$$

the ζ_j are the changepoints, satisfying

$$\min(x_n) \leq \zeta_1 \leq \zeta_2 \leq \dots \leq \zeta_j \leq \zeta_{j+1} \leq \dots \leq \zeta_{N_b} \leq \max(x_n) \quad (24)$$

and the B_j are the heights of the blocks.

The value of the observed quantity, y_n , at x_n , under this model is

$$\begin{aligned} \hat{y}_n &= \int w_n(x) B(x) dx \\ &= \int w_n(x) \sum_{j=1}^{N_b} B^{(j)}(x) dx \\ &= \sum_{j=1}^{N_b} \int w_n(x) B^{(j)}(x) dx \\ &= \sum_{j=1}^{N_b} B_j \int_{\zeta_j}^{\zeta_{j+1}} w_n(x) dx \end{aligned} \quad (25)$$

so we can write

$$\hat{y}_n = \sum_{j=1}^{N_b} B_j G_j(n) \quad (26)$$

where

$$G_j(n) \equiv \int_{\zeta_j}^{\zeta_{j+1}} w_n(x) dx \quad (27)$$

is the inner product of the n -th weight function with the support of the j -th block. The analysis in [?] shows how to deal with the non-orthogonality that is generally the case here.⁶

3.4 The Posterior

The averaging process in this data model induces dependence among the blocks. The likelihood, written as a product of likelihoods of the assumed independent data samples, is

$$P(\text{Data}|\text{Model}) = \prod_{n=1}^N P(y_n|\text{Model}) \quad (28)$$

$$= \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{1}{2}\left(\frac{y_n - \hat{y}_n}{\sigma_n}\right)^2} \quad (29)$$

⁶If the weighting functions are delta functions, it is easy to see that $G_j(n)$ is non-zero if and only if x_n lies in block j , and since the blocks do not overlap the product $G_j(n)G_k(n)$ is zero for $j \neq k$, yielding orthogonality, $\sum_N G_j(n)G_k(n) = \delta_{j,k}$. And of course there can be some orthogonal blocks, for which there happens to be no “spill over”, but these are exceptions.

$$= \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{1}{2}\left(\frac{y_n - \sum_{j=1}^{N_b} B_j G_j(n)}{\sigma_n}\right)^2} \quad (30)$$

$$= Q e^{-\frac{1}{2}\left(\frac{y_n - \sum_{j=1}^{N_b} B_j G_j(n)}{\sigma_n}\right)^2}, \quad (31)$$

where

$$Q \equiv \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma_n^2}}. \quad (32)$$

After more algebra and adopting a new notation, symbolized by

$$\frac{y_n}{\sigma_n^2} \rightarrow y_n \quad (33)$$

and

$$\frac{G_k(n)}{\sigma_n^2} \rightarrow G_k(n), \quad (34)$$

we arrive at

$$\log P(\{y_n\} | B) = Q e^{-\frac{H}{2}}, \quad (35)$$

where

$$H \equiv \sum_{n=1}^N y_n^2 - 2 \sum_{j=1}^{N_b} B_j \sum_{n=1}^N y_n G_j(n) + \sum_{j=1}^{N_b} \sum_{k=1}^{N_b} B_j B_k \sum_{n=1}^N G_j(n) G_k(n). \quad (36)$$

The last two equations are equivalent to Eqs. (3.2) and (3.3) of [?], so that the orthogonalization of the basis functions and the final expressions follow exactly as in that reference.

Often the measurement is made over a range of values of t , not just at a point. An good example is the spatial power spectra computed from measurements of the cosmic microwave background radiation, where the different experiments have widely different window functions (the term used to describe sensitivity as a function of the independent variable – *i.e.*, spatial harmonic number). In this case we have

$$x = \{x(t_n), t_n, w_n(t - t_n)\} \quad n = 1, 2, \dots, N, \quad (37)$$

where w gives the shape of the window function. This is a nontrivial complication if the window functions overlap, but can nevertheless be handled with the same technique.

3.5 Gaps and Mixed Data Modes

In some cases there are subintervals over which no events are possible (*e.g.* gaps due to failures in the detector system). What matters is the “live time” during the block, and this is simply the sum of the cell lengths. Thus data gaps can be handled by ignoring them! The only subtlety lies in interpreting what the model implies if a block extends across a gap. For each block the procedure yields the optimum rate parameter for whatever data lies in the block, ignoring any gaps. At the end of the procedure, for display purposes the gaps can be restored and plotted, preferably with some indication that rates within gaps are more uncertain.

Only if the fitness function depends on the total length of the block, and not just the live time, do the lengths of the overlap between the block and these gaps need to be included. The only example of this we have encountered results from the adoption of a prior distribution of block width.

Furthermore, one can even mix data modes. *E.g.*, bins of arbitrary sizes can be combined with point data. As with gaps the only burden for doing this is placed on the fitness function, which in this case would have to include a provision for data of mixed modes falling within the block. An example of this would be the analysis of both binned and time-tagged event (TTE) data for gamma-ray bursts observed by BATSE.

3.6 Prior for Number of Blocks

In our earlier work (Paper V) no explicit prior probability was assigned to N_{blocks} , the number of blocks (or equivalently the number of change-points). This omission amounts to using a flat prior, but in many contexts it is unreasonable to assign the same prior probability to all values. In particular, in most settings $N_{blocks} \approx N$ is *a priori* much more unlikely than is a small number of blocks.

For this reason it is desirable to impose a prior that assigns smaller probability to a large number of changepoints. Using a *geometric prior* for this parameter [Coram 2002] amounts to the prior

$$P(N_{blocks}) = P_0 \gamma^{-N_{blocks}} . \quad (38)$$

This is not the only prior possible, but it is very convenient to implement, since with the fitness equal to the log of the posterior, one only needs to subtract the constant $\log \gamma$ from the fitness of each block.

Figure 2 is the result of a simulation study using BATSE TTE data. The block decomposition of the full set of photons was taken as the (relative) truth and compared with decompositions based on a random subsample of the events. The RMS difference was taken as the measure of error, as a function of $\log \gamma$ (the abscissa in the figure). This procedure is analogous to standard *crossvalidation* methods. The effect⁷ of $\log \gamma$ in this and other simulations seems to level off at around 6. We have adopted the value 8 in the examples shown here. We recommend that persons using the algorithm carry out simulations of this kind to study the behavior of the algorithm as a function of γ for their application.

⁷A large value of this parameter naturally has the effect of reducing the number of blocks, producing a block representation that has less structure – giving a smoother visual appearance. But the parameter is not explicitly a smoothing parameter.

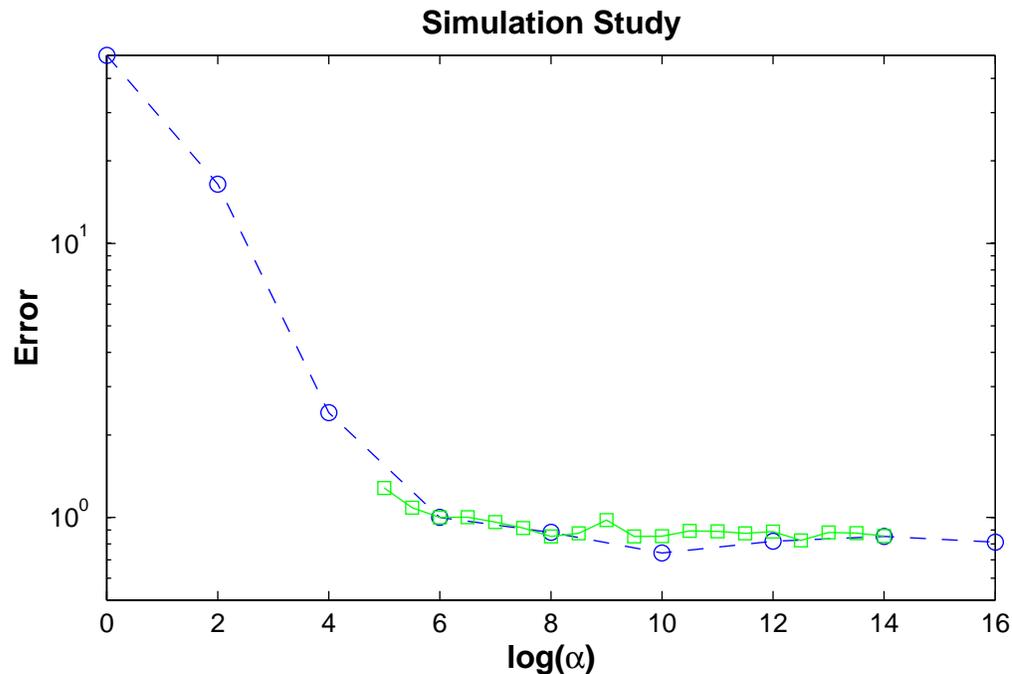


Figure 2: Simulation study for the parameter γ . The dashed curve with circles is for the first 110 bursts in the TTE sample, and the solid green curve with squares is for the complete sample of 1320 bursts, with DISCSC data.

4 Examples

This section presents results using the algorithms given in the Appendix on various sample data sets.

4.1 TTE Time Series

This algorithm was originally developed with the BATSE TTE data in mind. Paper V used the greedy approach, which not only is not guaranteed to achieve the global optimum, but the iterative process that implements the greedy optimization requires a stopping criterion based on the adoption of a threshold. Even though it is possible to choose well-justified values for the threshold, this nevertheless repre-

sents an undesirable ambiguity. It is one of the nice features of the current algorithm that there is no such threshold.

Figure 3 shows the optimal block decompositions of data for a γ -ray

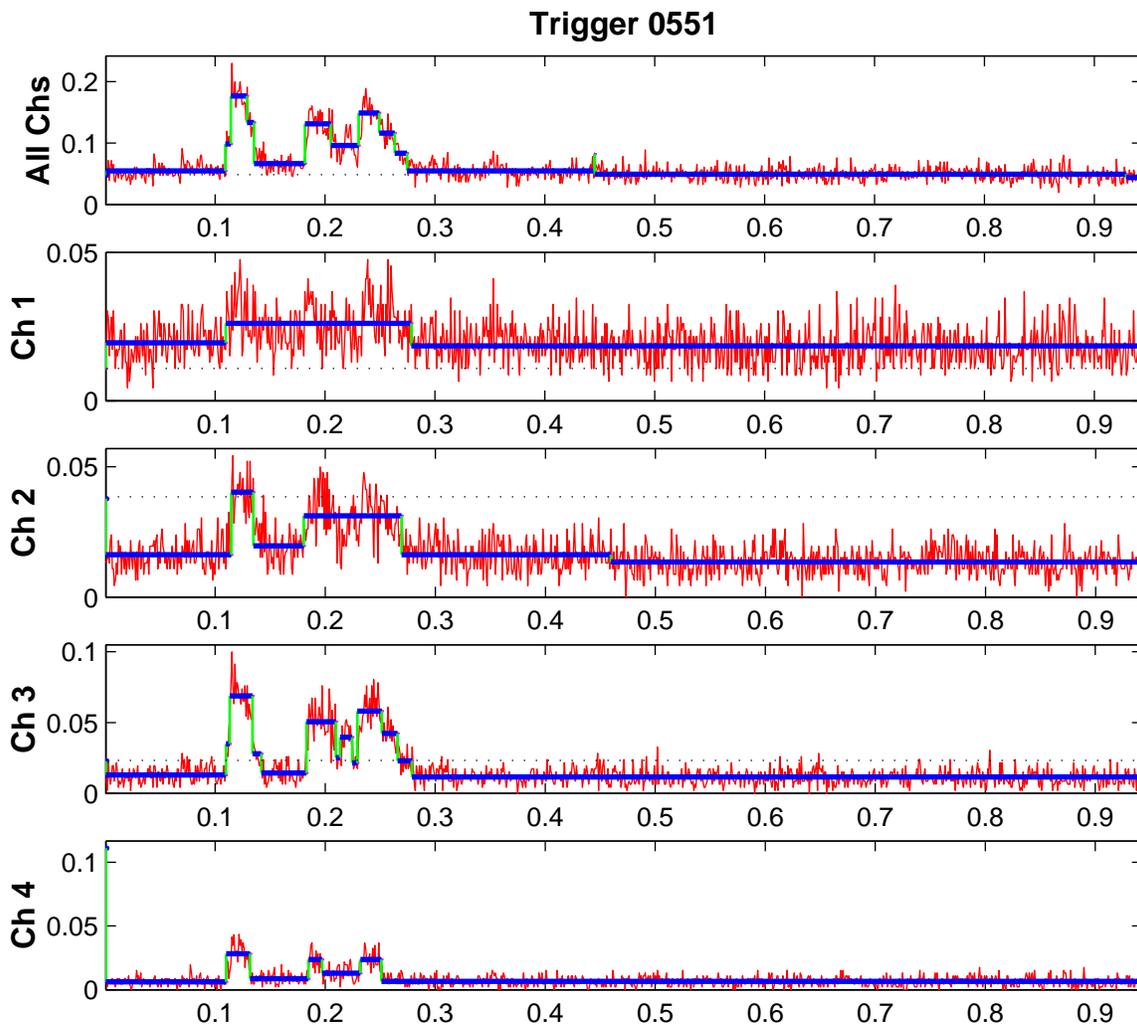


Figure 3: Optimal partitions of BATSE TTE data for Trigger 0551. All photons were used in the top panel; the others are based on the smaller number of photons detected in each of the four BATSE energy channels.

burst based on the point data comprising the TTE data for BATSE trigger 0551 (reference). The value $\log(\gamma) = 8$ was used for the parameter in the prior. This analysis is based on the first 14,000 photon time

tags for this burst. The full data set consists of 28,904 photons, but the last half is essentially background. Since the data are time tagged events, we used the form of the posterior given in Equation (12).

Figure 4 shows the TTE data summed over all four energy channels,

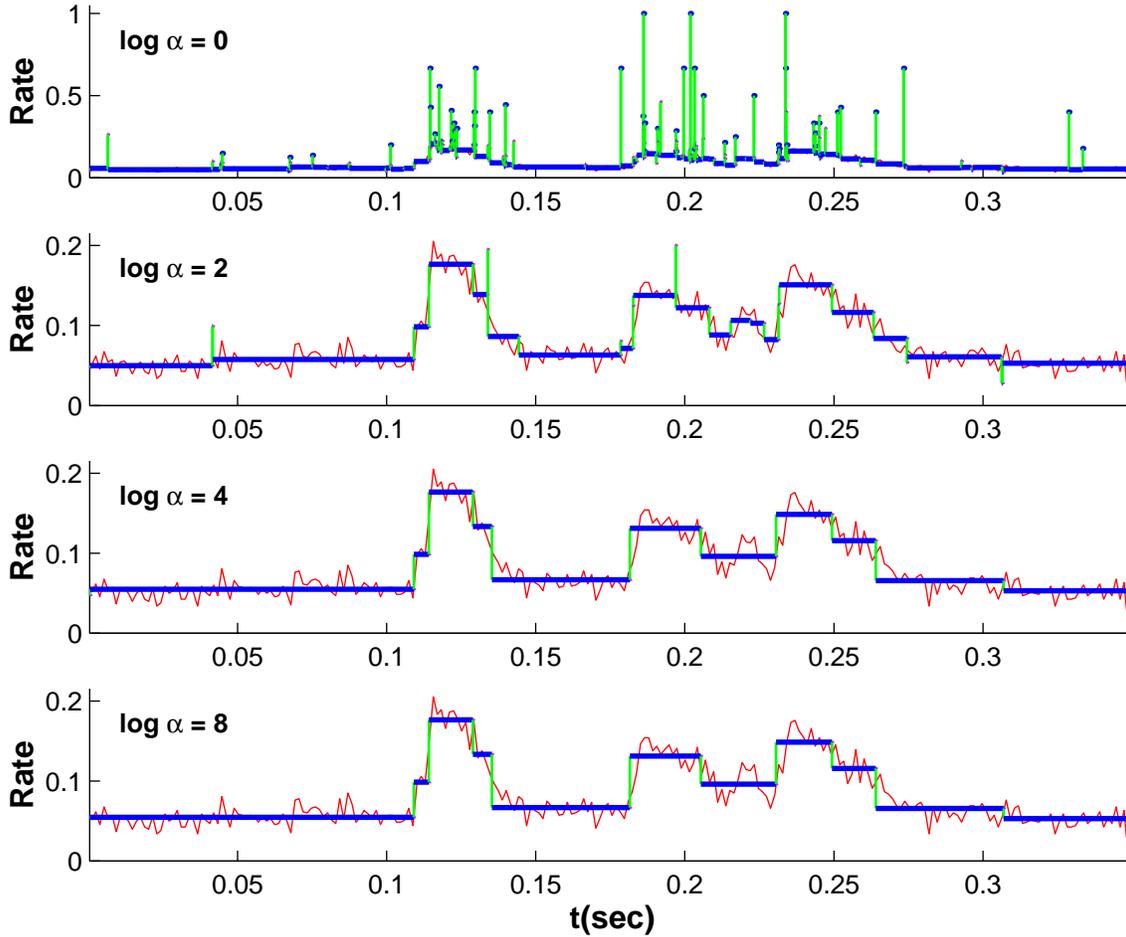


Figure 4: Optimal partitions of BATSE TTE data for Trigger 0551. Same as the first panel of Figure 3, except that four different values of $\log \gamma$ were used: 0, 2, 4 and 8

analyzed with four different values of the prior parameter $\log \gamma$. The first panel corresponds to a flat prior, giving too much prior probability to large numbers of changepoints. The obvious symptom is the appearance of many short spikes, corresponding to narrow intervals in which a statistical fluctuation is elevated by the inappropriate prior into ap-

parent significance. While they represent putative features that are probably not real, and are cosmetically obnoxious, these spikes would not affect any parameters derived essentially from the area under the curve.

The second panel, with a prior that gives lower weight to large numbers of changepoints has fewer spikes. By the time one reaches $\log \gamma = 4$, there is little change in the representation (*cf.* Figure 6). This result is not necessarily universal, but the figures shown here indicate that the value $\log \gamma = 8$ is quite reasonable and that values somewhat lower or higher would not make any real difference in the final representation.

4.2 Binned Time Series

Figure 5 shows the block representation for a portion of the light curve of the first burst in the BATSE catalog, observed on April 21, 1991, Trigger 0105. These data are available at

`ftp://coss.c.gsfc.nasa.gov/compton/data/batse/ascii_data/64ms/trig00000/cat64ms.00105`

in binned ⁸ format, with larger bins at the beginning, transitioning to smaller bins at the fiducial trigger time.

The three panels in the figure are for different values of the prior parameter $\log \gamma$. The first case, $\log \gamma = 0$, corresponds to a flat prior. With this rather strong encouragement for a large number of blocks, it is seen that the block representation is identical to the raw binned data. Even the coarse pre-trigger bins that seem to be combined into

⁸BATSE continuously recorded data in time bins 1.024 seconds long, and the time series posted on the web has 116 seconds of such low-resolution data pre-pended to the $16\times$ higher (64 millisecond bins) resolution data starting at the fiducial trigger time. To make the bins equal, the numbers given on the web site apportion the counts in each large bin into 16 small bins. Since our analysis can handle unequal bins, we have undone this, and reconstructed the actual integer counts in the larger bins.

large blocks because their event rates are so similar, are represented as separate blocks.

The second panel, $\log \gamma = 8$, corresponds to the best choice for the parameter, and can here be taken as the best block representation of these data. The last panel, $\log \gamma = 16$, corresponds to too much of a penalty against a large number of blocks. One notes that the most intense peak, which is resolved into two peaks in the other panels, is here a single peak.

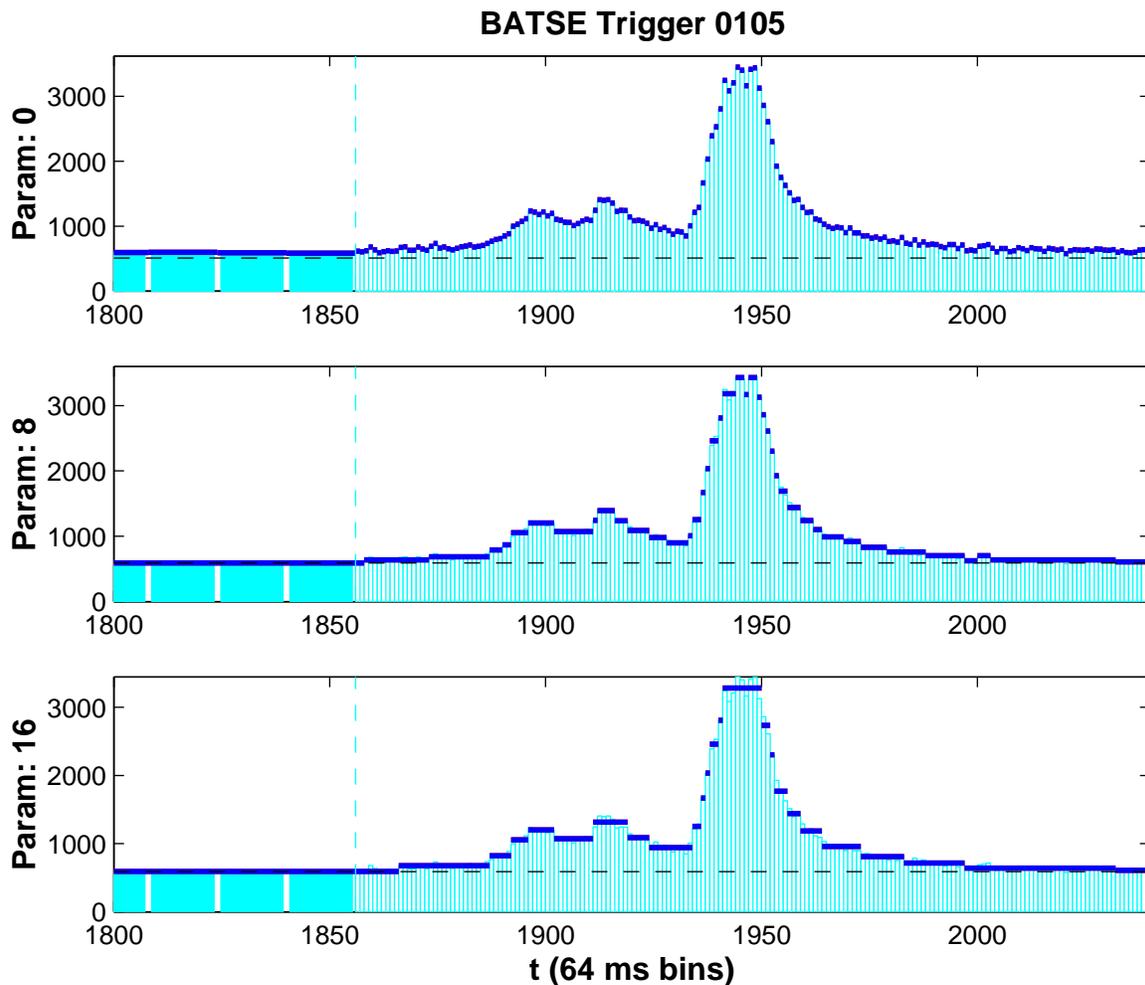


Figure 5: Optimal partitions of BATSE TTE data for Trigger 0105.

Finally, for comparison in Figure 6, we show analyses of the same

data, unbinned and binned, for Trigger 0551. This figure was created

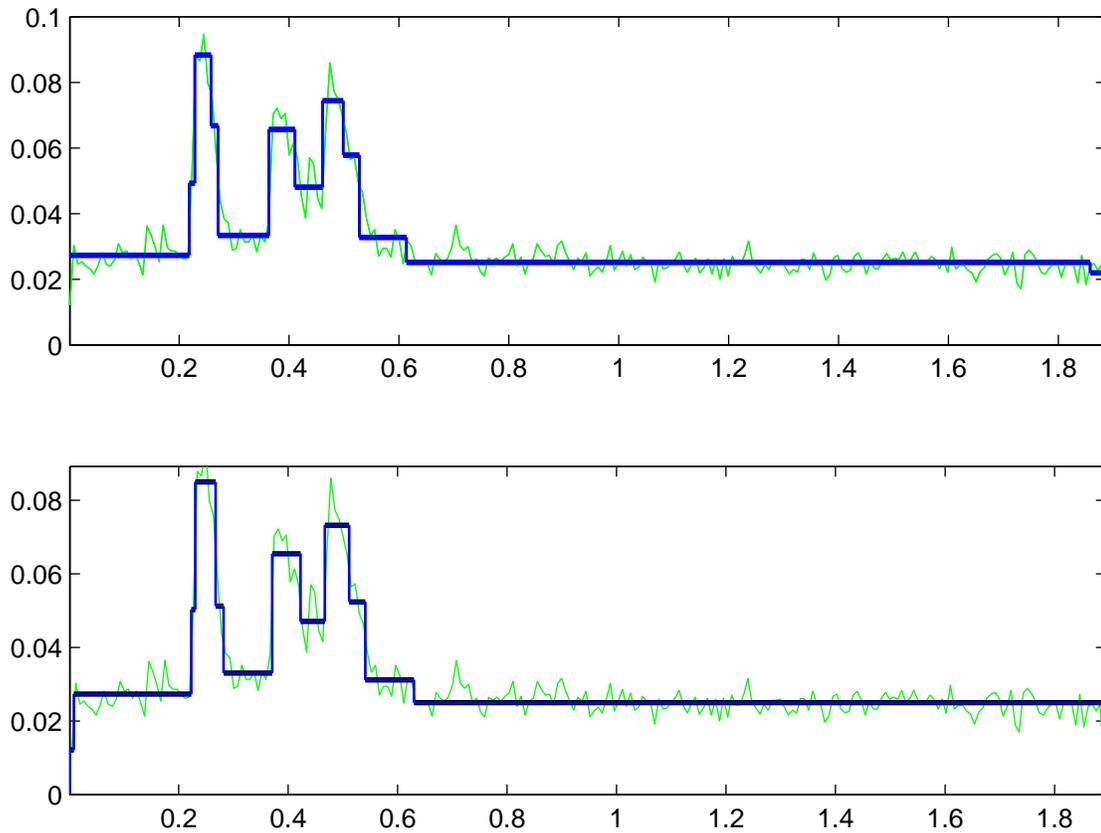


Figure 6: BATSE Trigger 0551. Top: TTE data. Bottom: same data, binned into 256 bins.

with the MatLab code included in the Appendix and available electronically. Note that the results are nearly identical, except for details of the first pulse.

4.3 Real Time Analysis

4.3.1 Triggers

4.3.2 Running Block Analysis

4.4 Histograms

5 Appendix A: MatLab Code

This section contains MatLab⁹ code for the analysis tools. The function `fit` evaluates the natural logarithm of the fitness function, and `reverse` reverses the order of an array. The quantity `eps` is the smallest number representable on the current machine. All other constructs and functions are standard MatLab.

5.1 Main Program

These code listings can be used to recreate Figure 6.

```
% test_global.m
%-----
% Run two test cases: (1) TTE data
%                   (2) binned data
%-----

first = 0; % Retrospective (not real-time) mode
tick2sec = .000002;

%-----
%   Load BATSE TTE Data
%-----

file_name = 'tteascii.00551';
tt = load_new_ttedata( file_name );
min_tt = min( tt );
max_tt = max( tt );
num_bins = 256;
bins = linspace( min_tt, max_tt, num_bins );
dt_bins = bins(2) - bins(1); % bin width
xx = hist( tt, bins );

%-----
%           Global Optimization of TTE Data
%-----

max_delt = 1; % Max separation of time tags
```

⁹© the Mathworks, Inc.

```

type = 1;
data_cells = make_data_cells( tt', type, max_delt );
prior = 8;

cpu_1_tte = cputime;
cpv_tte = global_optimum( data_cells, type, prior, first );
cpu_2_tte = cputime;

subplot(2,1,1)

% plot binned data for reference
plot( tick2sec * bins, xx/dt_bins, '-g') hold on
[ ii_pulses, count_vec ] = plot_tte( tt, tt( cpv_tte ) );
v = axis;
v(1) = tick2sec*min_tt;
v(2) = tick2sec*max_tt;
axis(v)

fprintf(1,'Done with TTE data; %4.2f cpu minutes.\n', (cpu_2_tte - cpu_1_tte)/60 )

%-----
% block analysis of same data, but binned
% cell_sizes is array of bin widths
% (here, all the same, but unequal in general)
%-----

cell_sizes = ( bins(2) - bins(1) ) * ones( size( xx ) );
data_cells = [ cell_sizes; xx; ]';

type = 2; % binned data
cpu_1_bin = cputime;
cpv_bin = global_optimum( data_cells, type, prior, first );
cpu_2_bin = cputime;

subplot(2,1,2)

% Set up data structure for plotting routine
cell_begin = min_tt + cumsum( cell_sizes ) - cell_sizes(1);
min_height = plot_partition( cpv_bin, xx, cell_sizes, cell_begin, xx );

fprintf(1,'Binned data; %4.3f cpu seconds.\n', (cpu_2_bin - cpu_1_bin) )

```

5.2 Construct Data Cells

```

function data_cells = make_data_cells( tt, type, max_delt, bin_size )
%-----
% Make data cells from time-tagged data - for input to global optimization
%
% Input:      tt -- array of time tags
%
%             max_delt -- maximum separation of times
%                   dt <= max_delt: in same cell
%                   dt > max_delt: in different cells
%
%             type -- cell type: 1: midpoints (~Voronoi)
%                               2: intervals (~Delaunay)
%                               3: bins
%
%             (dt = 0 corresponds to duplicate time tags)
%
% Output: cell_pops -- array of cell populations
%
%          cell_sizes -- array of cell sizes
%
% NB: length of type 2 output is one smaller than of type 1
%-----

if type == 3

    % binned data

    [ aa, bb ] = size( tt );
    if aa > bb; tt = tt'; end
    [ bin_flag, num_data ] = size( tt ); % force row vectors

    if bin_flag == 1
        cell_pops = tt;
        cell_sizes = bin_size * ones(size(tt));
    elseif bin_flag == 2
        cell_pops = tt(1,:); % bin populations
        cell_sizes = tt(2,:); % bins sizes
    else
        error('Incorrect matrix dimensions in global_optimum.m ...')
    end

else

```

```

% TTE data

cell_pops = ones( size( tt ) ); % Initial: one datum per cell

%-----
% Find clumps of points closer together than max_delt
%-----

ii_close = find( diff( tt ) < max_delt );

while ~isempty( ii_close )

    ii_start = ii_close(1); % Beginning of clump

    %-----
    % Index of end of the clump:
    % all ii_close-indices up to but not including
    % ii_beyond are in clump
    %-----

    ii_beyond = find( diff( ii_close ) > 1 );

    if isempty( ii_beyond )
        % All remaining close points are in the clump
        ii_end = ii_start + length( ii_close );
    else
        ii_end = ii_start + ii_beyond(1);
    end

    ii_clump = ii_start:ii_end;

    clump_pop = sum( cell_pops( ii_clump ) );
    clump_tt = mean( tt( ii_clump ) );

    % put members of the clump in one cell:
    cell_pops( ii_start ) = clump_pop;
    tt( ii_start ) = clump_tt;

    % null the cells evacuated by this operation:
    for ind = ii_end:-1:ii_start + 1

        cell_pops( ind ) = [];
        tt( ind ) = [];

    end

end

```

```
        ii_close = find( diff( tt ) < max_delt );

    end

    if type == 1

        %-----
        % midpoints
        %-----

        dt = diff( tt );
        ndt = length( dt );

        cell_sizes = 0.5 * ( dt(1:ndt-1) + dt(2:ndt) );

        dt_left = dt(1);
        dt_rite = dt(ndt);

        cell_sizes = [ dt_left cell_sizes dt_rite ];

    elseif type == 2

        %-----
        % intervals
        %-----

        cell_pops( length( cell_pops ) ) = []; % last datum can't define cp!
        cell_sizes = diff( tt );

    end

end % if type

data_cells = [ cell_sizes; cell_pops ]';
```

5.3 Global Optimum

```

function [ change_points, last, best ] = global_optimum( data_cells, type, ncp_p, first )
%=====
%   Find the optimum partition of sequential data
%-----
%
%   Input: data_cells -- sequential data array, a N x M array:
%               * column index: the N data cells (each row is one cell)
%               *   row index: the M parameters to compute the cost function
%
%               type -- data type: 1 --> time tagged data
%                               2 --> binned data
%
%               ncp_p -- log of parameter in geometric prior for number of changepoints
%
%               first -- 0 --> normal "retrospective" mode; analyze all data
%                       >0 --> trigger mode; return at first sign of a change
%-----
%
%   Output: change_points -- array of change points (index values for input array)
%
%               last -- working array of indices\
%                               |-- for diagnostic purposes only
%               best -- working array of optima /
%-----

[ num_cells, num_parameters ] = size( data_cells );

best = []; % "best(R)" is the value of the optimum at iteration R
last = []; % "last(R)" is the index at which this optimum occurs

%-----
%   Start with the first datum (R=1);
%   add the next one at each iteration
%-----

for R = 1:num_cells

    if R == 1
        qq = data_cells(R:-1:1, :);
    else
        qq = cumsum( data_cells(R:-1:1, : ) );
    end
end

```

```
end

[ best(R), last(R) ] = max( [0 best] + ...
    reverse( log_post( qq, ncp_p, type )' ) );

if first > 0 & last(R) > first
    % Trigger on first significant block
    change_points = last(R);
    return
end

end

%-----
% Find the optimum partition
%-----

index = last( num_cells );
change_points = [];

while index > 1

    change_points = [ index change_points ];
    index = last( index - 1 );

end
```

5.4 Load TTE Data

```

function [ times, channels, detectors ] = load_new_ttedata( file_name )
% load_new_ttedata.m
% Open and read data from TTE files.
% Strip off discrepancies at the end of the data.
% Return the times, channels, and detectors
%

% global talk
talk = 0;
%-----
%  Open the File
%-----
% file_in = ['../newdata/tteascii.0' file_name ]
file_in = [ file_name ];

if talk > 0
    fprintf(1,['Loading file ' file_in '\n' ])
end

[ fid message ] = fopen( file_in, 'r');

if fid == -1
    fprintf(1,['Error opening file ' file_in '\n'] )
else
    if talk > 0
        fprintf(1,['Successfully OPENED file ' file_in '\n'] )
    end
end

%-----
%  Read the File Header
%-----

format1 = '%s';

% Read the four "words" before the trigger ID
[ a1, count ] = fscanf(fid,format1,5);

% Read the trigger ID
[ a2, count ] = fscanf(fid,format1,1);

if talk > 0
    fprintf( 1,'File ID: %s \n', a2 )
end

```

```

% Read the two words before npts
[ a3, count ] = fscanf(fid,format1,3);

% Read npts
[ a4, count ] = fscanf(fid,'%f',1);

npts = a4;
% fprintf(1,'npts: %5.0f\n', npts)
if talk > 0
    fprintf( 1, 'npts: %10.0f\n', npts )
end

% read the remaining 23 words in the header of the file
[ a, count ] = fscanf(fid,format1,23);

%-----
% Now read the data: times, channels, detectors
%-----

[ times, count_times ] = fscanf(fid,'%f', npts);
[ channels, count_channels ] = fscanf(fid,'%f', npts);
[ detectors, count_detectors ] = fscanf(fid,'%f', inf);

%-----

if count_times ~= count_detectors
    fprintf(1, 'Different numbers of detectors!:\n')
    fprintf(1, 'Read %10.0f times.\n', count_times )
    fprintf(1, 'Read %10.0f channels.\n', count_channels )
    fprintf(1, 'Read %10.0f detectors.\n', count_detectors )
else
    if talk > 0
        fprintf(1, 'Read %10.0f times and channels.\n', count_times )
        fprintf(1, 'Read %10.0f times.\n', count_times )
        fprintf(1, 'Read %10.0f channels.\n', count_channels )
        fprintf(1, 'Read %10.0f detectors.\n', count_detectors )
    end
end

end

%-----
% carry out some checks
%-----

ii1 = find( channels ~= 1 & channels ~= 2 & channels ~= 3 & channels ~= 4);
if ~isempty(ii1) %% ~= []

```

```

    fprintf(1,'Warning: %4.0f channels are not 1,2,3 or 4!\n', length(ii1) )
end

ii2 = find( detectors ~= 0 & detectors ~= 1 & detectors ~= 2 & detectors ~= 3 ...
           & detectors ~= 4 & detectors ~= 5 & detectors ~= 6 & detectors ~= 7);
if ~isempty(ii2) % ~= []
    fprintf(1,'Warning: %4.0f detectors are not in 0-7!\n', length(ii2) )
end

[ a, count ] = fscanf(fid,format1,1);
if count == 0
    if talk > 0
        fprintf(1,'EOF\n')
    end
else
    fprintf(1,'Error; End of file not reached!\n')
end

%-----
% Close the File
%-----

message = fclose( fid );
if message ~= 0
    fprintf(1,['Error closing file ' file_in '\n'] )
end

%-----
% strip off last few points if they are discrepant, and rescale
%-----
max_strip = 10;
min_discrep = 10;

% Establish baseline of "good data"

n_times = length(times);
baseline_size = fix( 0.01 * n_times );

i2 = n_times - max_strip;
i1 = i2 - baseline_size + 1;
if i1 < 1,i1 = 1; end % Unlikely!

% Remove any points at the end that have a
% relative value that is much greater than
% the average value in the baseline region.

```

```
baseline = mean( times(i1:i2 ) );
discrep = ( times( n_times ) - baseline) / baseline;

count = 0;

while (discrep > min_discrep) & (count < max_strip)

    count = count + 1;
    n_times = length(times)-1;
    times = times(1:n_times);
    discrep = (times( n_times ) - baseline) / baseline;

end
%.....
n_times = length(times); % Just to be sure

    times = times(1:n_times);
    channels = channels(1:n_times);
    detectors = detectors(1:n_times);
```

5.5 Logarithm of the Posterior Probability

```

function log_prob = log_post( cell_data, ncp_prior, type )
%-----
% Log posterior (Bayes factor) for constant-rate Poisson data,
% * flat prior on Poisson rate parameter (unnormalized)
% * geometric prior on number of changepoints
%
% Input: cell_data -- MatLab structure containing these arrays:
%         cell_sizes -- size of each cell
%         cell_pops  -- number of events in each cell
%         ncp_prior  -- complexity parameter (from prior on number of changepoints)
%         type       -- 1 for TTE data; 2 for binned data
%
% Output: log_prob -- array of corresponding log-posterior probabilities
%-----

[ num_cells, num_arrays ] = size( cell_data );

cell_sizes = cell_data( :, 1 );
cell_pops  = cell_data( :, 2 );

if type == 1

    %-----
    %   TTE data
    %-----
    arg = cell_sizes - cell_pops + 1;

    ii = find( arg > 0 );
    num_bad = length( cell_sizes ) - length( ii );

    if num_bad == 0
        log_prob = gammaln( cell_pops + 1 ) + gammaln( arg ) ...
            - gammaln( cell_sizes + 2 );
    else
        log_prob = eps * ones( size( cell_pops ) ); % eps is smallest number

        log_prob(ii) = gammaln( cell_pops(ii) + 1 ) + gammaln( arg(ii) ) ...
            - gammaln( cell_sizes(ii) + 2 );
    end

elseif type == 2

```

```
% Binned data
log_prob = gammaln( cell_pops + 1 ) - ( cell_pops + 1 ) .* log( cell_sizes );

end

log_prob = log_prob - ncp_prior; % prior on number of changepoints
```

5.6 Plot partitions

```

function [ min_height, max_height ] = plot_partition( cpv, cell_pops, cell_sizes, cell_begi
% function [ min_height, max_height ] = plot_partition( cpv, cell_pops, cell_sizes, cell_begi
% Plots block representation for any set of cells,
% given the changepoints
% Input: cpv -- array of changepoint indices
% cell_pops -- array of cell populations (points in the cell)
% cell_sizes -- array of the width of the cells
% cell_begin -- array of beginning times of the cells
%             (needed to account for gaps in the data)
%             xx -- binned data (counts) so that xx / cell_sizes
%                 gives the event rate: points / unit time
% Output: min_height -- minimum block height
%         max_height -- maximum block height
%-----

small = 1.e-8;
tick2sec = .000002;

if isempty(cpv) | cpv(1) ~= 1
    % Make it so!
    cpv = [ 1 cpv ];
end

num_cells = length( xx );

tt_plot = tick2sec * ( cell_begin + 0.5*cell_sizes );
plot( tt_plot, xx ./ cell_sizes, '-g' )
hold on

% Sometimes last point is marked as a changepoint
while max( cpv ) >= num_cells
    cpv( length( cpv ) ) = [];
end

num_blocks = length( cpv );
min_height = 1e55;
max_height = -1e55;
yy_start = 0;

plot_sym_top = '-b';
line_width_top = 1;
plot_sym_side = '-b';
line_width_side = 1;

```

```

for id_block = 1:num_blocks

    index_start = cpv( id_block );

    if id_block == num_blocks
        index_end = num_cells;
    else
        index_end = cpv( id_block + 1 ) - 1;
    end

    ii_this = index_start:index_end;

    cell_pops_this = cell_pops( ii_this );
    cell_sizes_this = cell_sizes( ii_this );
    cell_begin_this = cell_begin( ii_this );

    t1 = min( cell_begin_this );
    t2 = max( cell_begin_this + cell_sizes_this );

    xx_this = xx( ii_this );
    if isempty( xx_this )
        xx_mean = 0;
    else
        xx_mean = sum( xx_this );
    end

    width_active = sum( cell_sizes_this );
    yy_plot = xx_mean / width_active;

    if yy_plot < min_height
        min_height = yy_plot;
    end
    if yy_plot > max_height
        max_height = yy_plot;
    end

    %-----
    % Horizontal line for whole block
    %-----

    hh = plot( tick2sec * [ t1 t2 ], yy_plot*[1 1], '-k' );
    hold on
    set(hh,'LineWidth', 2 )

    num_cells_this = length( cell_pops_this );

```

```

%-----
% Vertical line at block edge
%-----

t1 = cell_begin_this( 1 );
t2_that = t1;

hh = plot( tick2sec * [t1 t1], [ yy_start yy_plot ], plot_sym_side );
set(hh,'LineWidth', line_width_side )

yy_start = yy_plot;

for id_cell = 1:num_cells_this

    t1_this = cell_begin_this( id_cell );
    t2_this = t1_this + cell_sizes_this( id_cell );

    delt_tt = (t1_this - t2_that);
    if delt_tt > small

        % There is a gap between this and the previous cell
        % ... so plot the prior part of this block

        hh = plot( tick2sec * [t1 t2_that], yy_plot*[1 1], plot_sym_top );
        set(hh,'LineWidth', line_width_top )

        xx_patch = [ t2_that t1_this t1_this t2_that t2_that ];
        yy_patch = [ 0          0          yy_plot yy_plot  0          ];
        patch( tick2sec * xx_patch, yy_patch, 'g')
        disp('gap!')
        t1 = t1_this;

    end

    if id_cell == num_cells_this

        hh = plot( tick2sec * [t1 t2_this], yy_plot*[1 1], plot_sym_top );
        set(hh,'LineWidth', line_width_top )

    end

    t2_that = t2_this;

end

end
end

```

```
% do last vertical line
hh = plot( tick2sec * [t2 t2], [ yy_plot 0 ], plot_sym_side );
set(hh,'LineWidth', line_width_side )

v = axis;
v(1) = tick2sec* min( cell_begin );
v(2) = tick2sec * t2;
v(3) = 0;
v(4) = 1.05*max_height;
axis(v)
```

5.7 Plot TTE partitions

```

function [ ii_pulses, count_vec ] = plot_tte( event_times, change_times )
% function [ ii_pulses, count_vec ] = plot_tte( event_times, change_times )
% Input:  event_times -- raw times
%         change_times -- array of changepoint times
% Output: ii_pulses -- indices for the pulses (local maxima)
%         count_vec  -- block counts
%-----

tick2sec = 2.e-6;
plot_max = -Inf;

%-----
% Plot Blocks
%-----

num_cp = length( change_times );
num_blocks = num_cp + 1;
block_vec = [];
count_vec = [];
count_blocks = 0;

yy_old = 0;

if num_blocks > 0
for id = 0:num_cp

    if id < 1
        tick_left_this = event_times(1);
    else
        tick_left_this = change_times(id);
    end

    if id < num_cp
        tick_rite_this = change_times(id + 1);
    else
        tick_rite_this = event_times( length( event_times ) );
    end

    nn_this = length( find( event_times >= tick_left_this & ...
                            event_times <= tick_rite_this ) );

    delt_this = tick_rite_this - tick_left_this + 1;

    if delt_this > 0 % skip empty blocks

```

```

yy_plot = nn_this / delt_this;

count_blocks = count_blocks + 1;
block_vec = [ block_vec yy_plot ];
count_vec = [ count_vec nn_this ];

if id == 0
    yy_left = [ yy_plot yy_plot ];
else
    yy_left = [ yy_old yy_plot ];
end

xx_plot_left = tick2sec * tick_left_this;
xx_plot_rite = tick2sec * tick_rite_this;

%-----
% Left side
%-----
x_plt = [ xx_plot_left xx_plot_left ];
hh = plot( x_plt, yy_left, '-b');
set(hh,'LineWidth',1);

%-----
% Right side
%-----

% x_plt = [ xx_plot_rite xx_plot_rite ];
% hh = plot( x_plt, yy_plot, '-m');
% set(hh,'LineWidth',1);

%-----
% Top of Block
%-----

x_plt = [ xx_plot_left xx_plot_rite ];
hh = plot( x_plt, yy_plot*[1 1], '-b');
set(hh,'LineWidth',2)

if yy_plot > plot_max
    plot_max = yy_plot;
end

yy_old = yy_plot;

end

```

```
end
end

num_blocks = length( block_vec );

%-----
% Find peaks (local block maxima)
%-----
ii_left = 1:num_blocks-2;
ii_mid = ii_left + 1;
ii_rite = ii_left + 2;

ii_pulses = find( block_vec(ii_mid) > block_vec(ii_left) & ...
                 block_vec(ii_mid) > block_vec(ii_rite) );
```

5.8 Reverse (from *WaveLab*)

```
function r = reverse(x)
% reverse -- Reverse order of elements in 1-d signal
% Usage
%   r = reverse(x)
% Inputs
%   x      1-d signal
% Outputs
%   r      1-d time-reversed signal
%
% See Also
%   flipud, fliplr
%
%   r = x(length(x):-1:1);

%
% Copyright (c) 1993. David L. Donoho
%

%
% Part of WaveLab Version .701
% Built Tuesday, January 30, 1996 8:25:59 PM
% This is Copyrighted Material
% For Copying permissions see COPYING.m
% Comments? e-mail wavelab@playfair.stanford.edu
%
```

6 Bibliography

References

- [Coram 2002] Coram, Marc, (2002), personal communication and Ph. D. thesis, Nonparametric Bayesian Classification,
- <http://www-stat.stanford.edu/~mcoram/>
- [Donoho 1994] Donoho, D.L., (1994), Smooth Wavelet Decompositions with Blocky Coefficient Kernels, in *Recent Advances in Wavelet Analysis*, L Schumaker and G. Webb, eds., Academic Press, pp. 259-308.
- [Hubert, Arabie, and Meulman 2001] Hubert, L., Arabie, P., and Meulman, J., 2001, *Combinatorial Data Analysis: Optimization by Dynamic Programming*, SIAM: Philadelphia
- [Kolaczyk 1996] Kolaczyk, Eric D., (1996), "Estimation of Intensities of Inhomogeneous Poisson Processes Using Haar Wavelets," Technical Report 436, Department of Statistics, The University of Chicago, Chicago, to be submitted to *Journal of the Royal Statistical Society, Series B.*
- [Kolaczyk 1999] Kolaczyk, E.D. (1999). Bayesian Multi-Scale Models for Poisson Processes. *Journal of the American Statistical Association*, 94, 920-933.
- [Kolaczyk 2000] Kolaczyk, E.D. and Dixon, D.D. (2000). Nonparametric estimation of intensity maps using Haar wavelets and Poisson noise characteristics. *The Astrophysical Journal*, 534:1, 490-505.
- [Kolaczyk 1998] Kolaczyk, E.D. (1998) Wavelet Shrinkage Estimation of Certain Poisson Intensity Signals Using Corrected Thresholds. *Statistica Sinica*, 9, 119-135.
- [Kolaczyk 1998] Kolaczyk, E.D. (1997) Non-Parametric Estimation of Gamma-Ray Burst Intensities Using Haar Wavelets. *The Astrophysical Journal*, Vol. 483, 340-349.
- [Kolaczyk and Nowak 2002] Kolaczyk, Eric D. and Nowak, Robert D., (2002), "Multiscale Statistical Models," *Penn State Statistical Challenges* 3
- [Kolaczyk and Nowak 2002] Kolaczyk, Eric D. and Nowak, Robert D., (2002), "A Multiresolution Analysis for Likelihoods: Theory and Methods," preprint
- [Nowak and Figueiredo 2002] Nowak, Robert D., and Figueiredo, Mario A. T., "Unsupervised Progressive Parsing of Poisson Fields Using Minimum Description Length Criteria," preprint
- [Nowak and Figueiredo 2002] Nowak, Robert D., and Figueiredo, Mario A. T. (2002), "Unsupervised Segmentation of Poisson Data," preprint
- [Okabe, Boots, Sugihara and Chiu 2000] Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. (2000), *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, John Wiley and Sons, Ltd., New York, Second Edition
- [Ò Ruanaidh and Fitzgerald 1996] Ò Ruanaidh, J. J. & Fitzgerald, W. J., 1996, *Numerical Bayesian Methods Applied to Signal Processing*, Springer: New York.

- [Scargle 1981] Scargle, J. (1981), Studies in astronomical time series analysis. I: Modeling random processes in the time domain. *Ap. J. Supp.*, **45**, 1-71.
- [Scargle 1998] Scargle, J., 1998, "Studies in Astronomical Time Series Analysis. V. Bayesian Blocks, A New Method to Analyze Structure in Photon Counting Data", *Astrophysical Journal*, **504**, p. 405-418, Paper V. <http://xxx.lanl.gov/abs/astro-ph/9711233>
- [Scargle 2001a] Scargle, J. D., (2001), Bayesian Blocks: Divide and Conquer, MCMC, and Cell Coalescence Approaches, in *Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, 19th International Workshop, Boise, Idaho, 2-5 August, 1999. Eds. Josh Rychert, Gary Erickson and Ray Smith, AIP Conference Proceedings, Vol. 567, p. 245-256.
- [Scargle 2001b] Scargle, J. D., (2001a), "Bayesian Estimation of Time Series Lags and Structure," Contribution to **Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering (MAXENT 2001)**, held at Johns Hopkins University, Baltimore, MD USA on August 4-9, 2001.
- [Scargle 2001c] Scargle, J. D., (2001), "Bayesian Blocks in Two or More Dimensions: Image Segmentation and Cluster Analysis," Contribution to **Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering (MAXENT 2001)**, held at Johns Hopkins University, Baltimore, MD USA on August 4-9, 2001.
- [Scargle and Babu 2002] Scargle, J. D., and Babu, G. J. (2002), "Point Processes in Astronomy: Exciting Events in the Universe," Chapter 20 of *Handbook of Statistics: Stochastic Processes: Modeling and Simulation*, 2002, Elsevier Science.