Change point detection

- Find an abrupt change (in statistical properties) in time series data
- Applications in imaging (edge detection), human activity analysis, medical condition monitoring, etc
- Can be done online or offline
- Aminikhanghahi & Cook (2017) for an overview - see reading folder

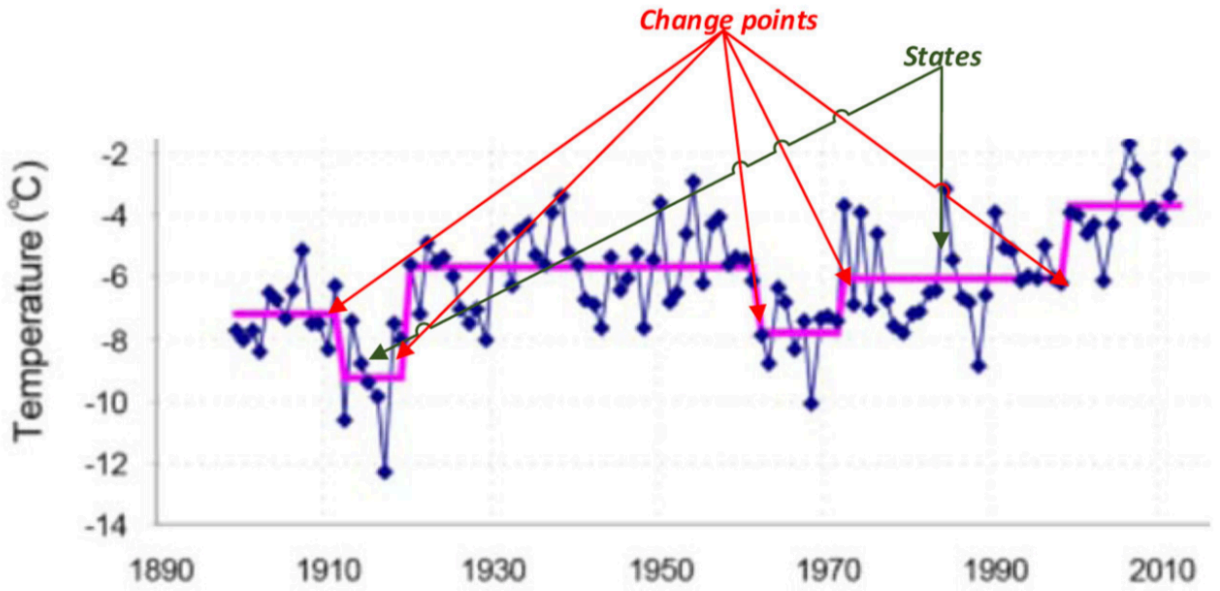
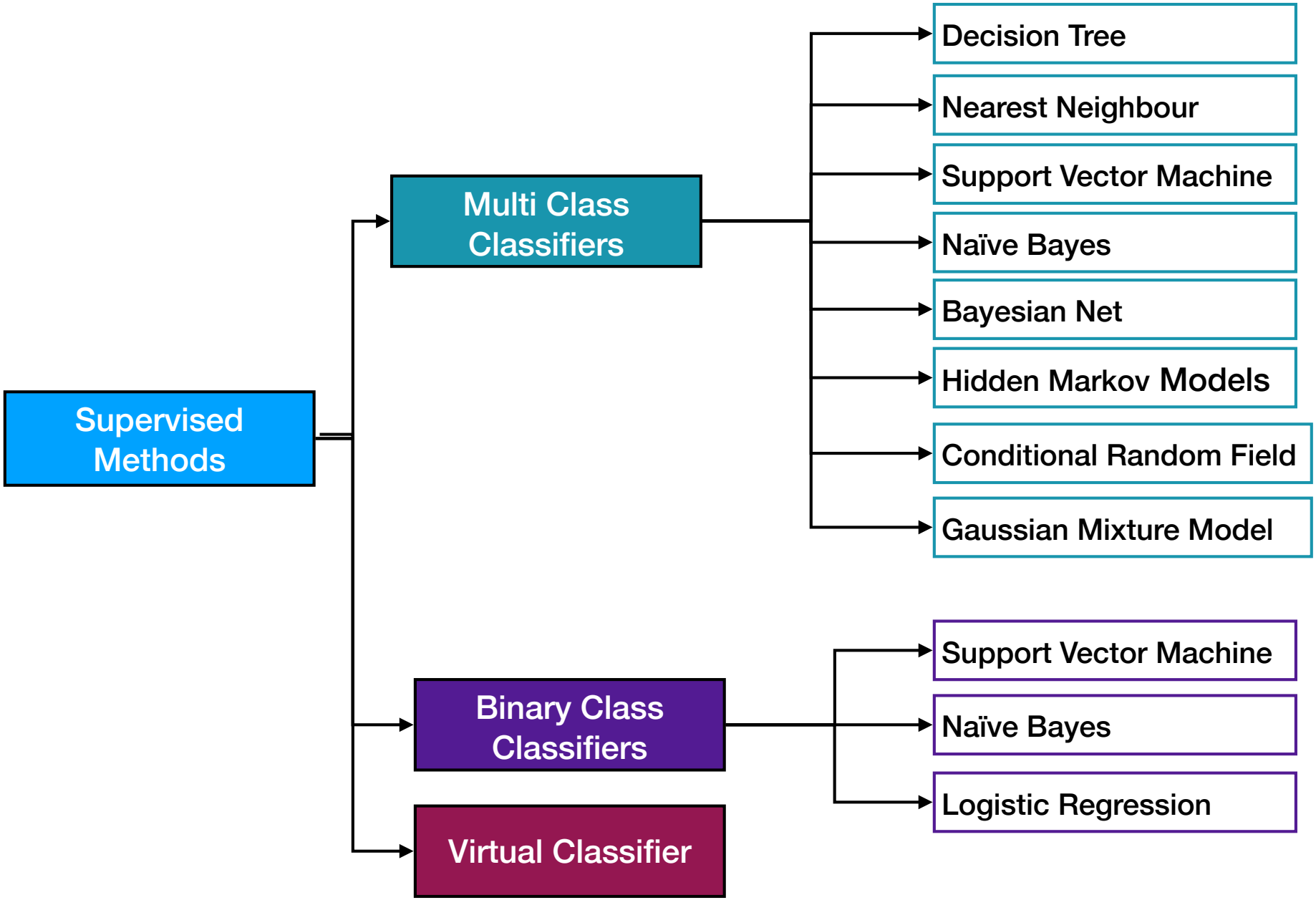


Figure 1.
Sample time series and change points (horizontal lines indicate separate states).

Change point detection

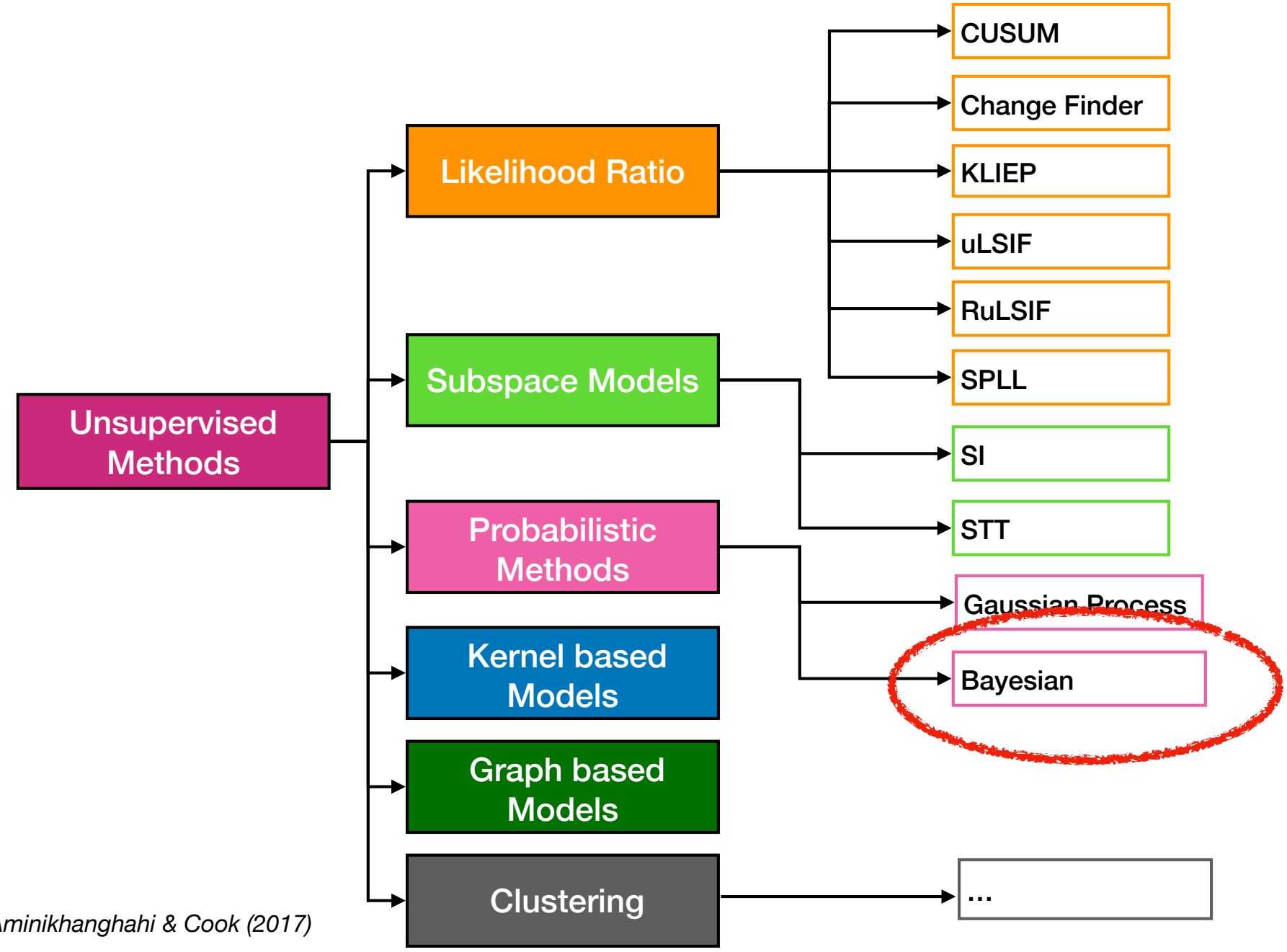
- Algorithms for change point detection
- Aminikhanghahi & Cook (2017) - mostly machine learning
- Supervised & Unsupervised methods

Change point detection



Aminikhanghahi & Cook (2017)

Change point detection



Aminikhanghahi & Cook (2017)

Change point detection

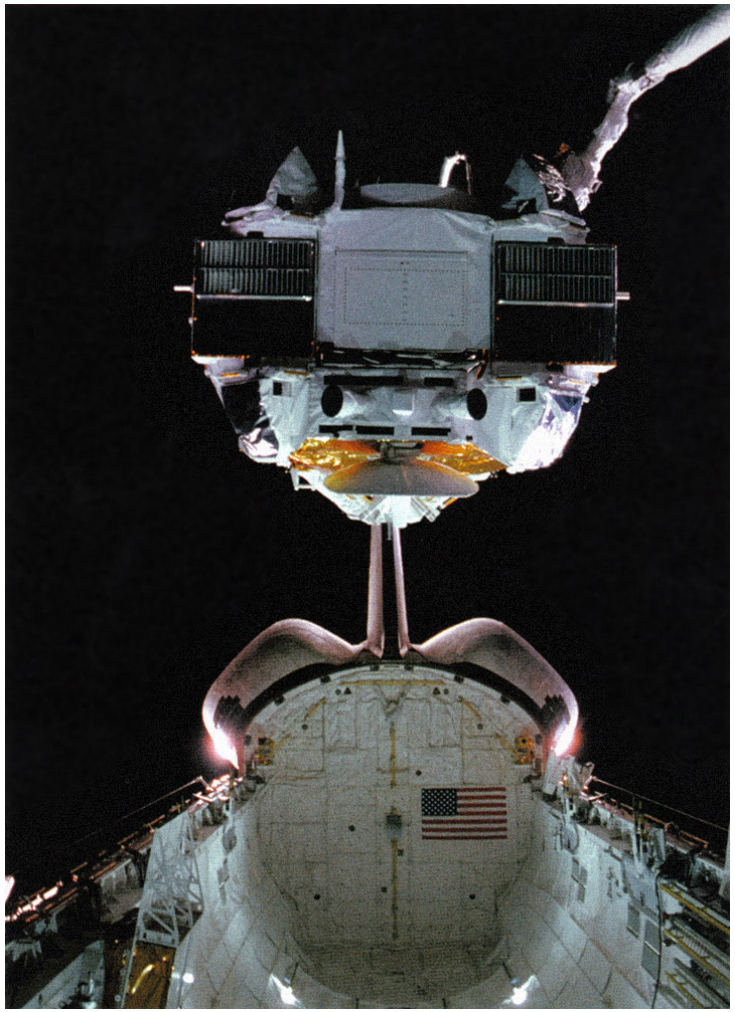
- We will explore a probabilistic method called Bayesian Blocks (Scargle 1998, Scargle et al. 2013)
- Segments the data into time intervals in which there is no statistically significant variations **within** the interval, but discontinuity **at** discrete points separating the intervals.
- Can be implemented in real-time (i.e. on a data stream) or offline

Bayesian blocks algorithm

- Goal is to find optimal segmentation of the data in the observation interval
- Non-parametric - limited assumptions about the data model
- Automatic penalty for model complexity
- Suppress observational errors while preserving valid information
- Can be use in ‘trigger’ mode, or offline

An aside: CGRO/BATSE

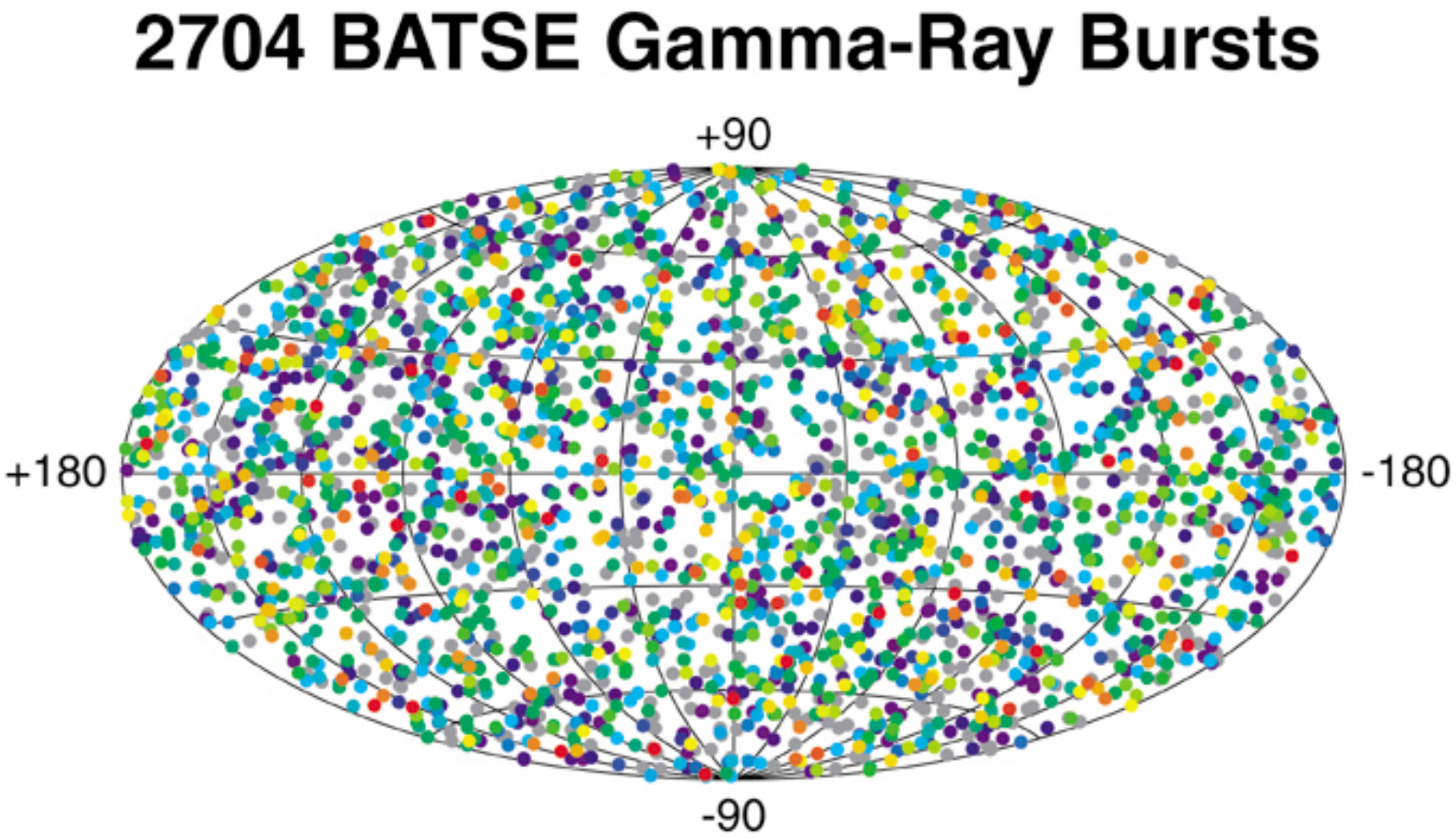
Burst and Transient Source Experiment



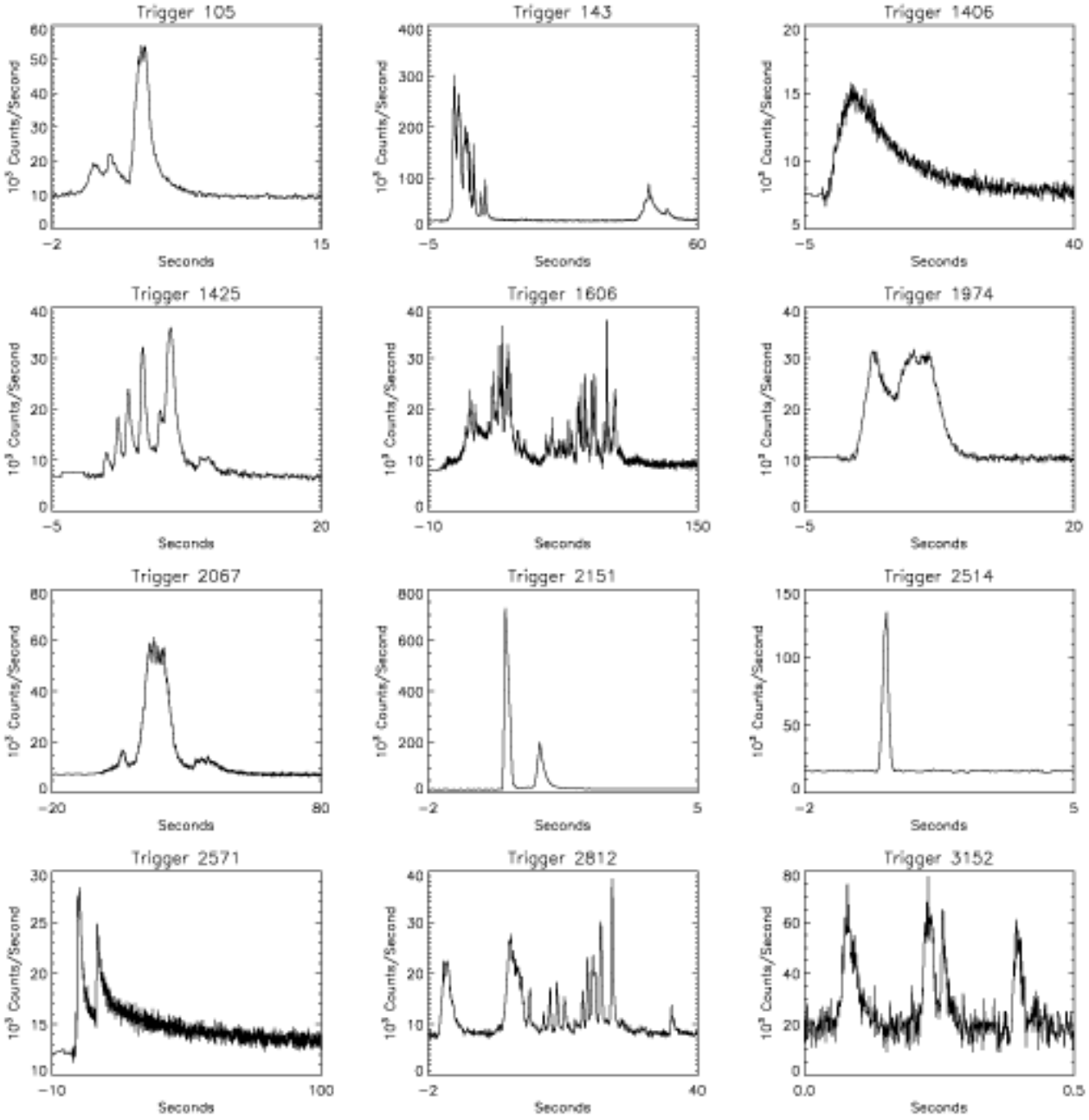
Launched 1991

Gamma-ray bursts

Discovered during the cold war



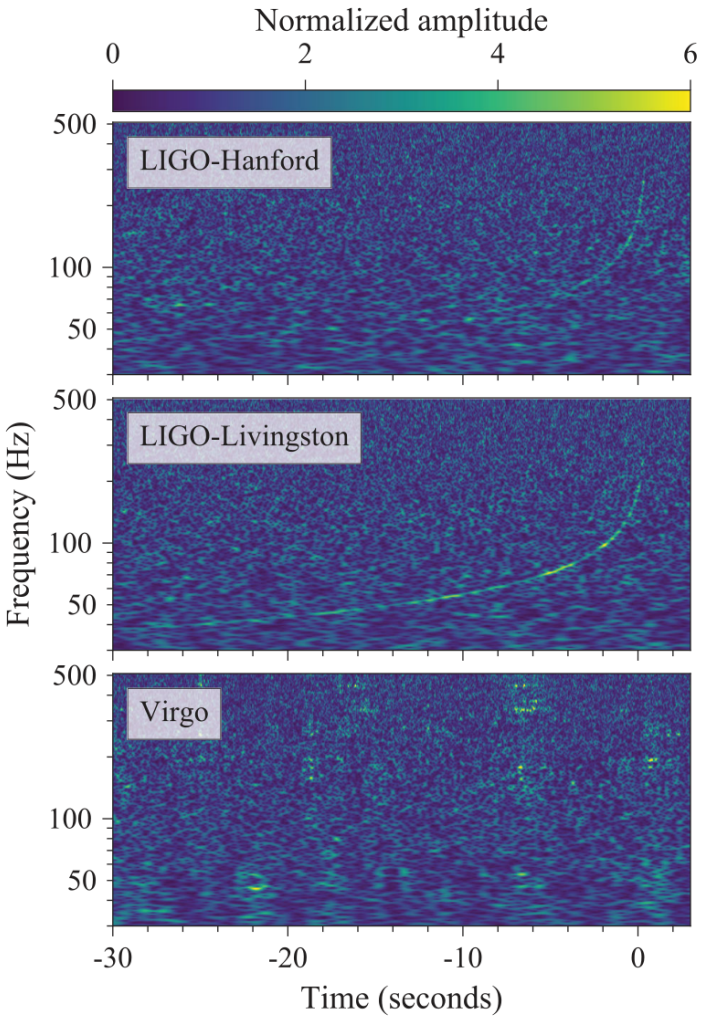
Gamma-ray bursts



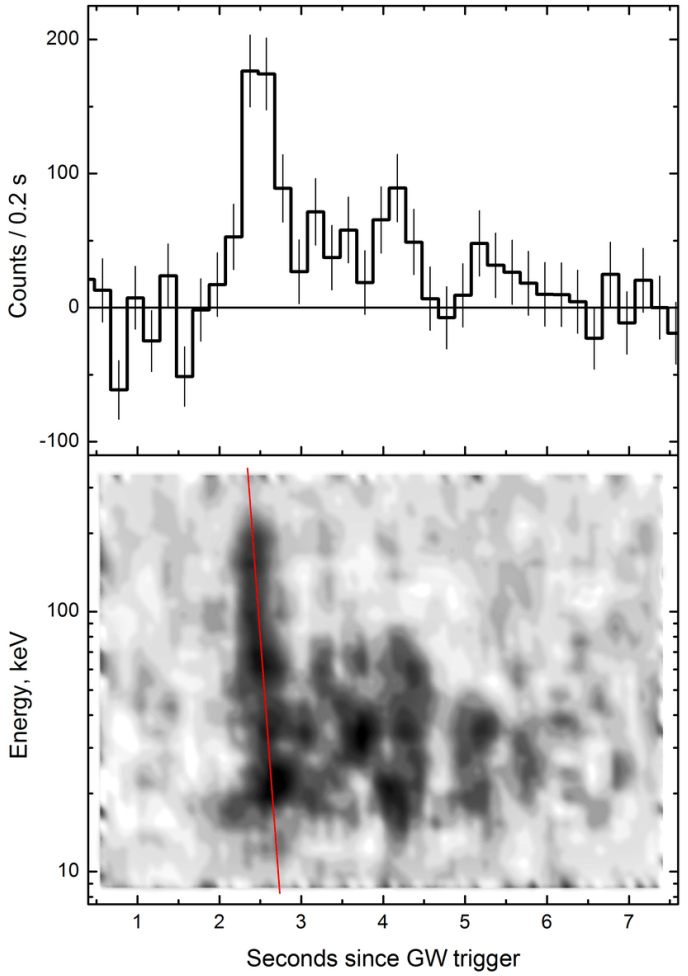
(Credit: J.T. Bonnell (NASA/GSFC))

The kilonova

GW170817: gravitational wave event



By LIGO Scientific Collaboration and Virgo Collaboration -



Associated with short gamma-ray burst GRB 170817A

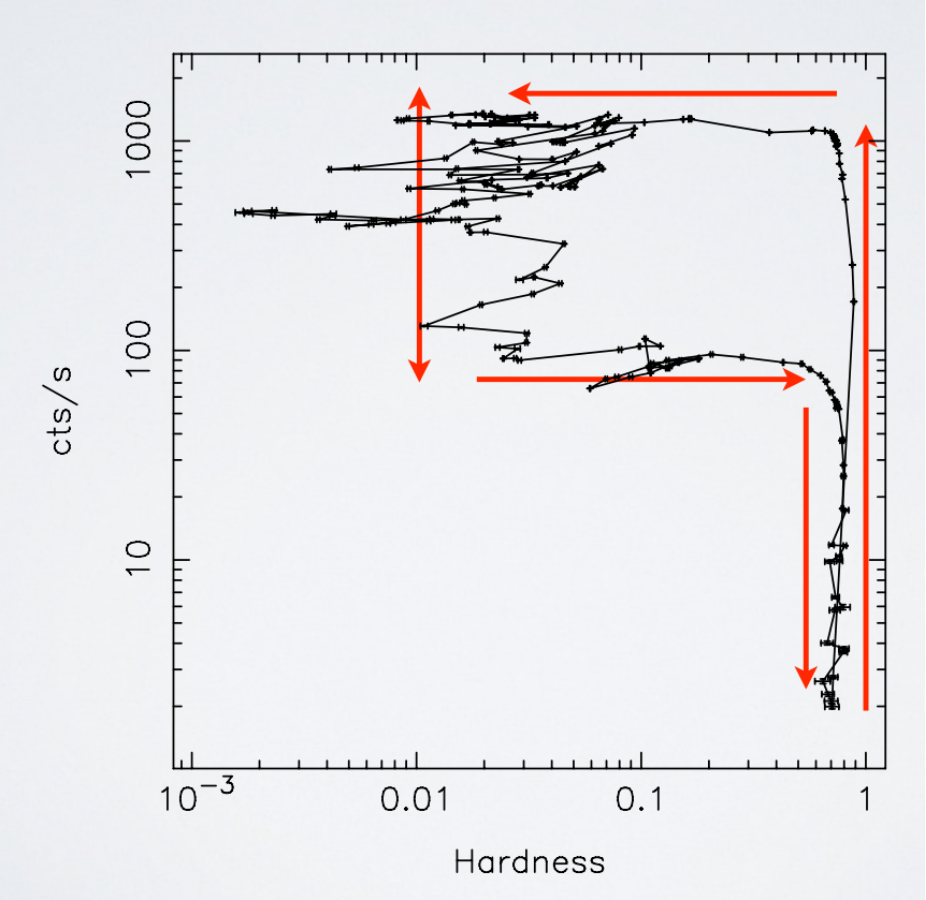
Pozanenko et al 2017

The kilo nova, double neutron star merger

Back to Bayesian blocks

- **Gamma-ray bursts**
- **State changes in black hole X-ray binaries**
- **Other applications?**

More detail in Scargle et al 2013 - see Reading folder on Vula



Jeroen Homan

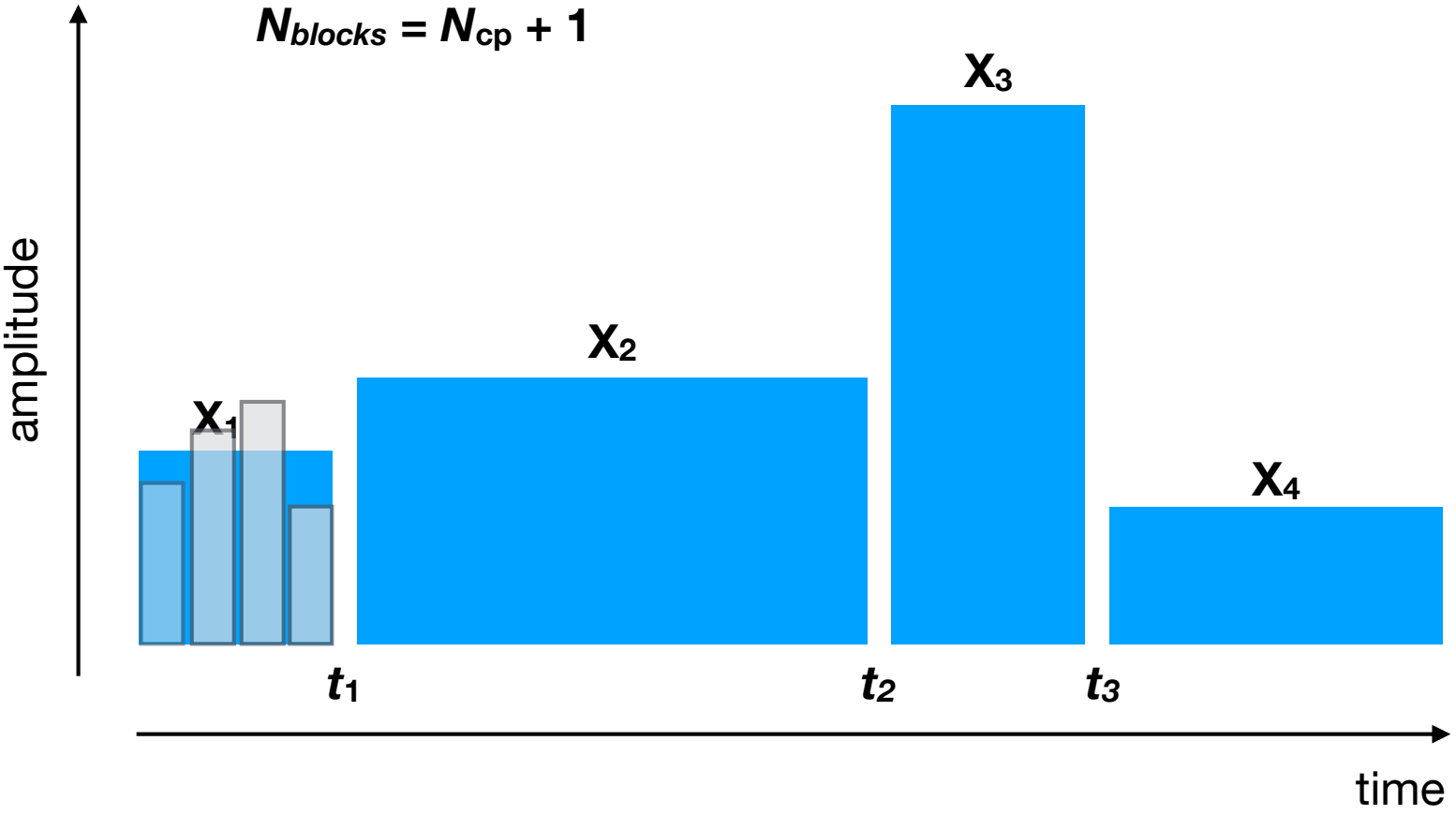
Piecewise constant model

$N_{cp} = 3$

$k = 1, 2, \ldots N_{cp} + 1$

$N_{blocks} = N_{cp} + 1$

N = number of data points, events



Piecewise constant model

- How many blocks?
- Define a prior distribution for the number of blocks
- Adjust a parameter controlling the steepness of this prior means we can push the algorithm towards either smaller or larger numbers of blocks.
- Prior becomes more important as S/N in the blocks decreases
- Don’t be weird about blocky interpretations of data - they’re useful.

Data modes & gaps

1. Times of events (TTE - time tagged events) Gamma-rays
2. Counts of events in time bins
3. Measurement of a quasi-continuous observable at some times radio flux measurements

For event data, correct for the *live time* during the block.

Data gaps can be handled by ignoring them.

Prior for the number of blocks

- Impose a prior that assigns a smaller probability to a large number of blocks
- Geometric prior (Coram 2002)

$$P(N_{\text{blocks}}) = P_0 \gamma^{N_{\text{blocks}}} \qquad 0 \leq N_{\text{blocks}} \leq N$$

$$P(N_{\text{blocks}}) = \frac{1-\gamma}{1-\gamma^{N+1}} \gamma^{N_{\text{blocks}}}$$

Taking gamma γ < 0.1 ensures the prior

Prior for the number of blocks

$$\langle N_{\text{blocks}} \rangle = P_0 \sum_{N_{\text{blocks}}=0}^{N} N_{\text{blocks}} \gamma^{N_{\text{blocks}}}$$

$$= \frac{N\gamma^{N+1} + 1}{\gamma^{N+1} - 1} + \frac{1}{1 - \gamma}$$

Estimated number of blocks is discontinuous, monotonic function of γ, but can't really use it to force a specific number of blocks.

You can think of γ as a free parameter than can be varied to adjust the amount of structure in the block representation.

Optimal segmentation of data on an interval

- Partitions $P(I) \equiv N_{\text{blocks}}; n_k, k = 1,2,3,...N_{\text{blocks}}$

  non-overlapping blocks, defined by number of blocks and block edges

- Data cell - a measurement

  needs time information, and some other information from which the fitness of a block can be calculated

- Fitness - something to optimize.

  any convenient measure of how well a constant signal represents the data in the block.

Fitness functions

- the measure of how well a constant signal fits the data in the proposed block
- needs to be comparable to other blocks
- different functions for *event data*, binned data and point measurements with normal errors (§3 Scargle et al 2013)

Fitness functions

Event data:

$$\log L(\theta) = \sum_n \log M(t_n, \theta) - \int M(t, \theta) dt$$

We use unbinned log-likelihood, see Scargle 2013 for explanation and reference. *M(t,θ)* is a model of the time dependence of the signal.

What is our block model?

M(t, λ) = λ i.e. a constant.

So,for block *k*:

$$\log L^{(k)}(\lambda) = N^{(k)} \log \lambda - \lambda T^{(k)}$$

where $N^{(k)}$ is the number of events in block *k* and $T^{(k)}$ is the length of the block.

The max of this likelihood is at $\lambda = N^{(k)}/T^{(k)}$ which gives…

Fitness functions

Event data:

$$\log L_{\max}^{(k)} + N^{(k)} = N^{(k)}(\log N^{(k)} - \log T^{(k)})$$

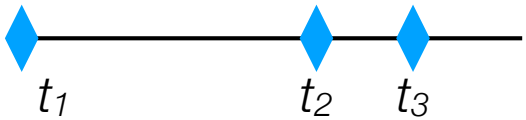
$N^{(k)}$ is the number of events in block k, and $T^{(k)}$ is the length of the block.

Robust in changing the time units

How do you define the length of the block for event data?

$$\Delta t_n = (t_{n+1} - t_{n-1})/2$$

This interval contains all times closer to event n than to any other event.
c.f. Voronoi tessellation



ncp_prior

Event data:

You can think about the prior as a “penalty” term.

From simulations of signal-free observational noise:

- generate synthetic pure noise time series

- apply BB for a range of ncp_prior

- select the smallest value that yields false positive rate of the change point below some value, *p0*

- generalize to get the probability of n change points

For event data:

$$\text{ncp\_prior} = 4 - 73.53\, p_0\, N^{-0.478}$$

Note the mistake in Scargle 2012!!!

Algorithm §2.6 Scargle+ 2013

Begin with the first data cell.
At each step, one more cell is added.

Let $P^{opt}(R)$ denote the optimal partition of the first R cells.

In the starting case $R = 1$, the only option is that there is one block consisting of the first cell.

Moving on from this, we have identified the optimal partition at step R ($P_{opt}(R)$), and have saved the value of the optimal fitness in array **best**, location of last change point in **last**.

How do we get $P^{opt}(R+1)$?

For some r consider the set of all partitions (of these first R + 1 cells) whose last block starts with cell r.
The fitness of this block: $F(r)$

Algorithm §2.6 Scargle+ 2013

For some *r* consider the set of all partitions (of these first R + 1 cells) whose last block starts with cell *r*.
The fitness of this block: *F(R)*

*T*he only member of this set that could possibly be optimal is that consisting of $P^{opt}(r-1)$ followed by this last block.

The fitness of this partition is the sum of *F(R)* and the fitness of $P^{opt}(r-1)$ saved from previous step:

$$A(r) = F(r) + \begin{cases} 0 & r = 1 \\ \text{best}(r-1) & r = 2,3,...,R+1 \end{cases}$$

Over the indicated range of *r* the above expresses the fitness of all the partitions *P(R+1)* that can possibly be optimal.

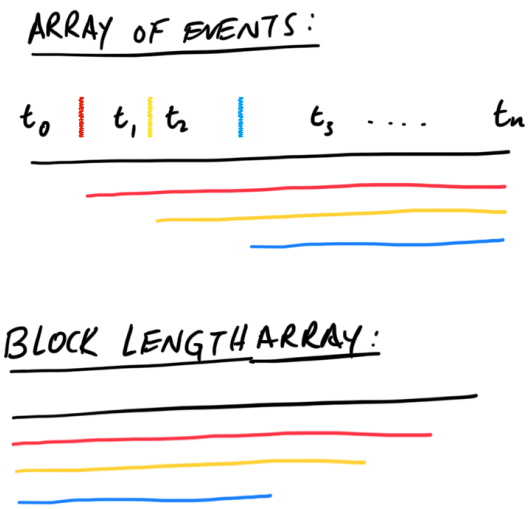
Hence the value of *r* yielding the optimal partition $P^{opt}(R+1)$ is easily computed by maximising *A(R)*.

Algorithm

Matlab code

```
% For data modes 1 and 2:
% nn_vec is the array of cell populations.
% Preliminary computation:
block_length=tt_stop-[tt_start 0.5*(tt(2:end)
 +tt(1:end-1))' tt_stop];
...
%-------------------------------------------
% Start with first data cell; add one cell at
   each iteration
%-------------------------------------------
best = [];
last = [];
for R = 1:num_points
   % Compute fit_vec: fitness of putative last
      block (end at R)
    if data_mode == 3% Measurements, normal
     errors
        sum_x_1 = cumsum(cell_data(R:-1:1,
         1))'; %sum(x/sig^2)
        sum_x_0 = cumsum(cell_data(R:-1:1,
         2))'; %sum(1/sig^2)
        fit_vec=((sum_x_1(R:-1:1)) .^ 2).
        /(4*sum_x_0(R:-1:1));
    else
        arg_log = block_length(1:R) - block_
         length(R+1);
        arg_log(find(arg_log <= 0))=Inf;
        nn_cum_vec=cumsum(nn_vec(R:-1:1));
        nn_cum_vec = nn_cum_vec(R:-1:1);
        fit_vec = nn_cum_vec .*(log(nn_cum_
         vec) - log(arg_log));
    end
    [best(R), last(R)] = max([0 best] +
      fit_vec - ncp_prior);
end
```

Cell edges



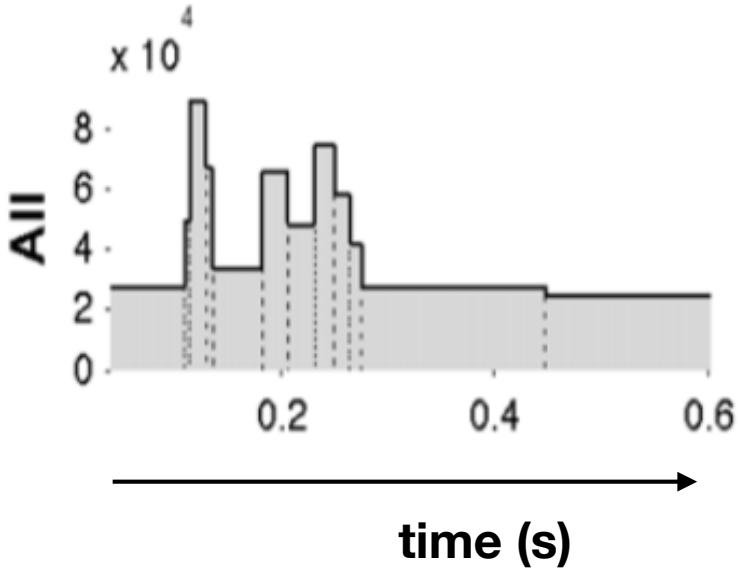
$$\log L_{\max}^{(k)} + N^{(k)} = N^{(k)}(\log N^{(k)} - \log T^{(k)})$$

Apply the prior, and maximise the fitness

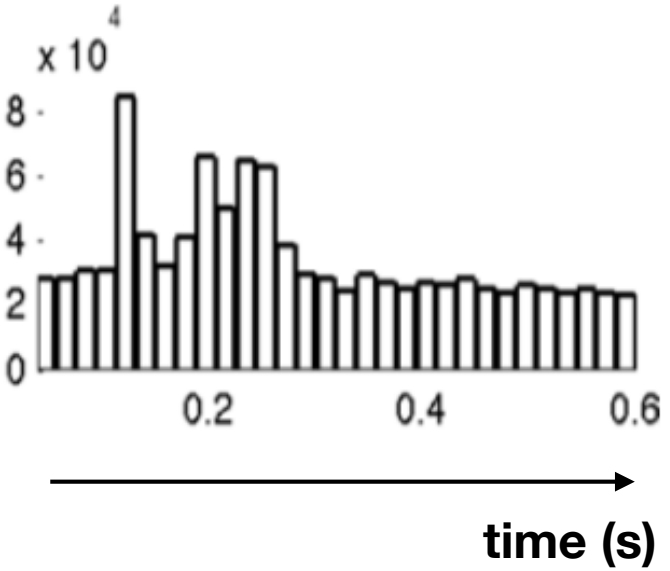
Algorithm

```
%-------------------------------------------
% Now find changepoints by iteratively peeling
   off the last block
%-------------------------------------------
index = last(num_points);
change_points = [];
while index > 1
    change_points = [index change_points];
    index = last(index - 1);
end
```

BATSE trigger 0551



Bayesian blocks



Evenly binned

For you to try

- Download the BATSE trigger 0551 data from Vula
- Implement the bayesian blocks technique (see e.g. astropy) to reproduce figure in previous slide
- How does the value of p_0 affect the blocks?
- Can you do better at estimating ncp_prior by running simulations?