

# Multi-dimensional Point Process Models in R

Roger D. Peng

*Department of Statistics, University of California, Los Angeles  
Los Angeles CA 90095-1554*

## Abstract

A software package for fitting and assessing multi-dimensional point process models using the R statistical computing environment is described. Methods of residual analysis based on random thinning are discussed and implemented. Features of the software are demonstrated using data on wildfire occurrences in Northern Los Angeles County, California.

## 1 Introduction

This paper introduces an R package for fitting and assessing multi-dimensional point process models. In particular, the software is designed for conducting likelihood analysis of conditional intensity models. While the methodology for applying maximum likelihood to point process models is already well-developed, techniques for assessing the absolute goodness-of-fit of models are still being actively researched. We describe several methods of *point process residual analysis* such as random rescaling, thinning, and approximate thinning, and discuss their advantages and disadvantages in practice. The package implements the two random thinning-based methods.

Section 2 begins with a brief discussion of the specification and fitting of point process conditional intensity models. Section 3 describes several residual analysis techniques and discusses their various properties. Section 4 describes the `ptproc` package. In particular, we show how features of the R language such as lexical scope and the ability to manipulate language objects are used to build a compact flexible framework for analyzing point process models. Finally, Section 5 shows two extended examples of how one might use the `ptproc` package.

The current version (1.0) of `ptproc` was written and has been tested with R version 1.6.0. The package is written in pure R code so it should be usable by anyone with access to an R interpreter.

## 2 Point Process Models

A point process can be thought of as a random measure  $N$  specifying the number of points,  $N(A)$ , in any compact set  $A \subset S$ , where  $S$  is simply the domain in which the point process resides. The measure is non-negative integer-valued and is finite on any finite subset of  $S$ . Usually, one assumes that the point process is *simple*, so that each point is distinct with probability one. For the purposes of this paper, we will only look at simple point processes in the space-time domain (i.e.  $S \subset \mathbb{R}^3$ ), with one time dimension and two spatial dimensions.

A simple point process is conveniently specified by its conditional intensity  $\lambda(t, \mathbf{x})$  ( $(t, \mathbf{x}) \in S$ ), defined as the limiting expectation,

$$\lim_{\substack{\Delta t \downarrow 0 \\ \Delta \mathbf{x} \downarrow 0}} \frac{1}{\Delta t \Delta \mathbf{x}} \mathbb{E} [N((t, t + \Delta t) \times (\mathbf{x}, \mathbf{x} + \Delta \mathbf{x})) \mid H_t]$$

where  $H_t$  represents the internal history of the process up to time  $t$  and  $N((t, t + \Delta t) \times (\mathbf{x}, \mathbf{x} + \Delta \mathbf{x}))$  is the number of points in a small neighborhood of  $(t, \mathbf{x})$ . When it exists, the conditional intensity can be interpreted as the instantaneous rate of occurrence of events at time  $t$  and location  $\mathbf{x}$ .

Since the conditional intensity completely specifies the finite dimensional distributions of the point process  $N$  (Daley and Vere-Jones, 1988), modelling  $N$  typically involves directly modelling  $\lambda$ . Many conditional intensity models have been developed for specific applications, most significantly from the field of seismology (e.g. Vere-Jones, 1970; Ogata, 1999) and also ecology (Rathbun and Cressie, 1994b).

## 2.1 Model Fitting

Given a parametrized model for the conditional intensity (with parameter vector  $\boldsymbol{\theta}$ ), the unknown parameters can be estimated by maximizing the log-likelihood function

$$\ell(\boldsymbol{\theta}) = \int_S \log \lambda(t, \mathbf{x}; \boldsymbol{\theta}) N(dt, d\mathbf{x}) - \int_S \lambda(t, \mathbf{x}; \boldsymbol{\theta}) dt d\mathbf{x} \quad (1)$$

It has been shown that under general conditions, the maximum likelihood estimates are consistent and asymptotically normal (Ogata, 1978; Rathbun and Cressie, 1994a; Rathbun, 1996). When maximizing the log-likelihood, care must be taken to ensure that  $\lambda$  is positive at all points in  $S$ . One solution to this potential difficulty is to model  $\log \lambda$  rather than model  $\lambda$  directly (see e.g. Berman and Turner, 1992). Another method is to include a penalty in the specification of the log-likelihood which penalizes against parameter values which produce negative values of the conditional intensity. For example, one could use the following modified log-likelihood function

$$\ell^*(\boldsymbol{\theta}) = \ell(\boldsymbol{\theta}) - P(\boldsymbol{\theta})$$

where  $P(\boldsymbol{\theta})$  is a suitable penalty function. For example, one could use a smooth penalty function as in Ogata (1983). An alternative would be simply to add a penalty any time the conditional intensity takes a negative value. Given an appropriate  $\alpha > 0$ , let

$$P(\boldsymbol{\theta}) = \alpha \mathbf{1}\{\lambda(t, \mathbf{x}; \boldsymbol{\theta}) < 0, (t, \mathbf{x}) \in S\} \quad (2)$$

where  $\mathbf{1}\{A\}$  is the indicator of the event  $A$ .

## 3 Residual Analysis Methods

A common method of evaluating a point process model is to examine likelihood criteria such as the Akaike Information Criterion or the Bayesian Information Criterion (e.g. Ogata, 1988; Ogata and Tanemura, 1984; Vere-Jones and Ozaki, 1982). These criteria provide useful numerical comparisons of the global fit of competing models. For example, a common model for comparison is the homogeneous Poisson model. However,

these criteria cannot shed light on the absolute goodness-of-fit of a particular model. In particular, they cannot identify where a model fits poorly and where it fits well.

Residual analysis in other statistical contexts (such as regression analysis) is a powerful tool for locating defects in the fitted model and for suggesting how the model should be improved. While the same is true in point process analysis, one must be careful in how one defines the residuals in this context. For point processes the “residuals” consist of yet another point process, called the *residual process*. There exist various ways of constructing a residual process and we discuss some of those methods in this Section.

The common element of residual analysis techniques is the construction of an approximate homogeneous Poisson process from the data points and an estimated conditional intensity function  $\hat{\lambda}$ . Suppose we observe a one-dimensional point process  $t_1, t_2, \dots, t_n$  with conditional intensity  $\lambda$  on an interval  $[0, T]$ . It is well known that the points

$$\tau_i = \int_0^{t_i} \lambda(s) ds \quad (3)$$

for  $i = 1, \dots, n$  form a homogeneous Poisson process of rate 1 on the interval  $[0, n]$ . This new point process is called the residual process. If the estimated model  $\hat{\lambda}$  is close to the true conditional intensity, then the residual process resulting from replacing  $\lambda$  with  $\hat{\lambda}$  in (3) should resemble closely a homogeneous Poisson process of rate 1. Ogata (1988) used this random rescaling method of residual analysis to assess the fit of one-dimensional point process models for earthquake occurrences. For the multi-dimensional case, Schoenberg (1999) demonstrated that for a large class of point processes, the domain can be rescaled in such a way that the resulting process is again homogeneous with rate 1.

When rescaling a multi-dimensional point process one may encounter two practical difficulties:

1. The boundary of the rescaled domain may be uninterpretable (or unreadable). That is, the residual process may be homogeneous Poisson but the irregularity of the rescaled domain can make the points difficult to examine. In particular, an irregular boundary can bias various tests for uniformity which are sensitive to edge effects.
2. Integration of the conditional intensity function is required. In practice, accurate integration of the conditional intensity in certain dimensions can be computationally intensive.

Both of these problems can be ameliorated by instead constructing a residual process via random thinning. Suppose that for all  $(t, \mathbf{x}) \in S$  there exists a value  $m$  such that

$$0 < m \leq \inf_{(t, \mathbf{x}) \in S} \lambda(t, \mathbf{x}). \quad (4)$$

Then for each  $i = 1, \dots, n$ , we delete the data point  $(t_i, \mathbf{x}_i)$  with probability  $1 - m/\lambda(t_i, \mathbf{x}_i)$ . The undeleted points form a homogeneous Poisson process of rate  $m$  over the original domain  $S$ . The residual process obtained through this method of thinning will be referred to as the *ordinary thinned residual process* or *ordinary thinned residuals*. For details on random thinning, see Lewis and Shedler (1979) and Ogata (1981).

To construct the ordinary thinned residual process, we only need to evaluate the conditional intensity. Typically, this is a much simpler task than integrating the conditional intensity. Unfortunately, in some cases where  $m$  is very close to zero, the thinning process can result in very few points. However, because of the randomness involved in the thinning procedure, one can repeat the thinning many times and examine the various realizations for homogeneity. Another drawback of this method is that there may not exist such an  $m$ .

A third method for constructing a residual process addresses the problem of having too few points in the thinned process. Each data point  $(t_i, \mathbf{x}_i)$  is assigned a probability  $p_i$  such that

$$p_i \propto \frac{1}{\lambda(t_i, \mathbf{x}_i)}.$$

Then, a weighted subsample of size  $K$  ( $< n$ ) is drawn from the vector  $\{(t_1, \mathbf{x}_1), \dots, (t_n, \mathbf{x}_n)\}$  using the weights  $p_1, \dots, p_n$ . As long as  $K$  is sufficiently small relative to  $n$ , the resulting set of points  $\{(t_1^*, \mathbf{x}_1^*), \dots, (t_K^*, \mathbf{x}_K^*)\}$  should resemble (approximately) a homogeneous Poisson process of rate  $K/|S|$  over the original domain. The process constructed using this method will be referred to as the *approximate thinned residual process*, or simply *approximate thinned residuals*. This procedure can also be used to generate many realizations of the residual process and each realization will have exactly  $K$  points. Approximate thinned residuals were used in Schoenberg (2002) to assess the space-time Epidemic-Type Aftershock Sequence model of Ogata (1998).

The residual process is useful for model evaluation purposes because the process is homogeneous Poisson if and only if the model is equal to the true conditional intensity function. Once the residual process has been constructed, it can be inspected (graphically) for uniformity and homogeneity. In addition, numerous statistical tests can be applied to test the process for uniformity (see e.g. Diggle, 1983; Ripley, 1979, and many others). For example, Ripley’s  $K$  function can be used to test for spatial clustering and inhibition (Ripley, 1976). In general, any deviation of the residual process from a homogeneous Poisson process can be interpreted as a deviation of the model from the true conditional intensity.

## 4 The ptproc Package

The `ptproc` source package and updates can be downloaded from the Comprehensive R Archive Network<sup>1</sup> (CRAN). The design of the package is influenced heavily by the Statistical Seismology Library (SSLIB) of Harte (1998). In particular our template for the conditional intensity function is similar to that of SSLIB. However, SSLIB is particularly focused on applications in seismology and can be awkward to apply to other types of data. Here we are attempting to provide a general framework for analyzing a variety of multi-dimensional point process data. Those who are interested in seismological applications are encouraged to look at SSLIB because of its many other useful functions related to earthquake analysis.

After installing the package it can be loaded into R in the usual way:

```
> library(ptproc)
Multi-dimensional Point Process Models in R (version 1.0)
>
```

### 4.1 The “ptproc” Object

The package introduces the class “`ptproc`” to represent a point process model and data object. A “`ptproc`” object contains the data points, the conditional intensity function, parameter values, and other information required for fitting and evaluating the conditional intensity. The `ptproc` package contains some built-in conditional intensity functions but in general, the user will want to specify code for evaluating and integrating the conditional intensity over the study area. Currently, the “`ptproc`” class has `print`, `summary`, `logLik`, and `residuals` methods associated with it.

The constructor function `ptproc` constructs the point process object. Required elements which must be passed as arguments are:

---

<sup>1</sup><http://cran.r-project.org>

- **pts**: A matrix of data points. The matrix should be  $n \times p$  where  $n$  is the number of data points and  $p$  is the number of dimensions in the data. If the user only wants to simulate a point process (and not fit one) then **pts** can be set to `NA`.
- **cond.int**: The conditional intensity function. This should be the name of an existing function (either in the global workspace or one of the example functions included in the package).
- **params**: Parameters for the model. The values specified here will be used as initial values when maximizing the log-likelihood function.

Other elements of the “**ptproc**” object include:

- **fixed.params**: A vector equal in length to **params** containing `NA` and non-`NA` values. A `NA` in the  $i$ th position of the **fixed.params** vector indicates that the  $i$ th parameter is a free parameter. If an element in the **fixed.params** vector is non-`NA`, then the corresponding parameter in **params** is fixed. By default, all parameters are set to be free (i.e. every element of **fixed.params** is `NA`).
- **initial.params**: After a model has been fit to data (via **ptproc.fit**) the initial values of the parameters are stored in this element.
- **condition**: An R expression. See Section 4.2 for details.
- **ranges**: A matrix containing upper and lower boundaries for each dimension of the data. The **ranges** element specifies the domain of the point process. The matrix will be a  $2 \times p$  matrix, where  $p$  is the number of dimensions in the data. The  $(1, j)$  element gives the lower bound in the  $j$ th dimension and  $(2, j)$  element gives the upper bound in the  $j$ th dimension. If **ranges** is not specified, the minima and maxima of the data points in each dimension are used.
- **data**: Other information (usually in the form of a list) that may be needed to evaluate or integrate the conditional intensity. This may include covariate data, point process marks, or preprocessed values. The default value for **data** is `NULL`.
- **ndim**: The dimensionality of the point process. This is only needed when one is simulating from a point process model and there is no data matrix from which to obtain the dimensionality.

The most important part of the “**ptproc**” object is the conditional intensity function. This is where the user has to write the most code. The form of the conditional intensity function should adhere to the following template.

1. The name of the function should be `[user-specified name].cond.int`. For example in Section 5 we will use a simple linear model and the name of the conditional intensity function is `linear.cond.int`.
2. The arguments to the conditional intensity function should be

```
my.cond.int <- function (params, eval.pts, pts = NA, data = NULL, TT = NULL)
```

where **params** is a vector of parameters for the model, **eval.pts** is a matrix of points at which we want to evaluate the conditional intensity, **pts** is the matrix containing the original data, **data** can be any other information that the conditional intensity function may need, and **TT** is a matrix denoting the ranges of integration in each dimension. The object passed through the **TT** argument should be of the same form as the **ranges** element of the “**ptproc**” object.

3. It is the responsibility of the user to make sure that the conditional intensity function can be evaluated at all of the data points (given a valid set of parameters) and that the integral over the entire domain can be evaluated. For fitting a model, it is not required that the conditional intensity be evaluated at *all* points in the domain; just the data points. However, for plotting and general visualization purposes, evaluation at all points will likely be useful.
4. The body of the conditional intensity function will generally appear as follows:

```
my.cond.int <- function (params, eval.pts, pts = NA, data = NULL, TT = NULL) {
  a <- params[1]
  b <- params[2]
  ## Assign other parameters

  if(is.null(TT)) {
    ##
    ## Evaluate the conditional intensity at eval.pts
    ##
  }
  else {
    ##
    ## Integrate the conditional intensity over the entire domain
    ##
  }
  ## return a value: either a vector of values equal to nrow(eval.pts)
  ## when evaluating the conditional intensity, or a single value when
  ## integrating.
}
```

See Appendix A for examples.

## 4.2 Maximizing the Log-Likelihood

The log-likelihood of a given conditional intensity model can be computed using the `logLik` method. This method simply computes value in (1) by summing the conditional intensity values at each data point and evaluating the integral over the entire domain.

The package function `ptproc.fit` is used for fitting the conditional intensity model via maximum likelihood. It first calls `make.optim.logLik` to construct the negative log-likelihood function which will be passed to the optimizer. In general, there is no need for the user to call `make.optim.logLik` directly. However, for models with a small number of parameters it may be useful for exploring the likelihood surface. The entire point process object is included in the environment of the objective function so that all of the data and parameters are accessible inside the optimizer. The scoping rules of R (Gentleman and Ihaka, 2000) make the implementation of this mechanism clean and fairly straightforward. The R function `optim` is then called to minimize the negative of the log-likelihood function. `optim` provides four optimization procedures: a quasi-Newton method of Broyden, Fletcher, Shanno, and Goldfarb, a conjugate gradient method, the simplex algorithm of Nelder and Mead (1965), and a simulated annealing procedure based on that of Belisle (1992). See also Nocedal and Wright (1999) for details on the first two optimization procedures. The user

must choose from these procedures based on the form of the conditional intensity model. For relatively smooth models with a few parameters, the quasi-Newton and conjugate gradient methods tend to produce good results. For models with many parameters, the simulated annealing method may be useful for obtaining a good initial solution. The default Nelder-Mead method tends to produce reasonable results for a wide class of models.

For the purposes of demonstration, we will fit a homogeneous Poisson model to some simulated data. This model prescribes a constant conditional intensity over the entire domain (which we will take to be  $[0, 1]^3$ ). That is,  $\lambda(t, \mathbf{x}) = \mu$  for all  $(t, \mathbf{x}) \in [0, 1]^3$ . We generate the data with the following commands:

```
> set.seed(1000)
> x <- cbind(runif(100), runif(100), runif(100))
```

The code for the conditional intensity function is as follows,

```
hPois.cond.int <- function(params, eval.pts, pts = NA, data = NULL, TT = NULL) {
  mu <- params[1]

  if(is.null(TT))
    rep(mu, nrow(eval.pts))
  else {
    vol <- prod(apply(TT, 2, diff))
    mu * vol
  }
}
```

Finally, we construct the point process object with

```
> ppm <- ptproc(pts = x, cond.int = hPois.cond.int, params = 50,
+             ranges = cbind(c(0,1), c(0,1), c(0,1)))
```

For this example, 50 was chosen as the initial parameter value for the optimization procedure.

After constructing a “`ptproc`” object with the `ptproc` function, one can attempt to fit the model using `ptproc.fit`. The command

```
> fit <- ptproc.fit(ppm, method = "BFGS")
```

would minimize the negative log-likelihood using the BFGS quasi-Newton method. Tuning parameters can be passed as a (named) list to `optim` via the argument `optim.control`. For example, it may be desirable to see some tracing information while fitting the model. One can do this by running

```
> fit <- ptproc.fit(ppm, optim.control = list(trace = 2), method = "BFGS")
```

instead. The values in the `params` element of the “`ptproc`” object are used as the initial parameters for the optimizer. Certain parameters can be held fixed by setting appropriated values in the `fixed.params` vector.

As it was discussed in Section 2.1, it may be necessary to modify the log-likelihood function to include a penalty term. Normally, including a penalty into the evaluation of the log-likelihood would involve directly modifying the code for the log-likelihood function. Then for each model which required a different type of penalty, the log-likelihood function would have to re-written. The `ptproc` package takes a different approach, which is to store the penalty term with the particular model object. These user-defined penalties are then

evaluated when the log-likelihood is evaluated. This way, the penalty is identified with the model rather than the log-likelihood function.

The `condition` element is included in the “`ptproc`” object for the purpose of including penalties. By default, it is set to `NULL`. However, one can include a penalty by using the `penalty` function. For example, suppose we wanted to penalize the log-likelihood for negative values of any of the parameters. We could set

```
> condition(ppm) <- penalty(code = NULL, condition = "any(params < 0)")
```

The `penalty` function returns an unevaluated R expression and the function `condition` modifies the “`ptproc`” object so that the R expression is included. When `ppm` is passed to `ptproc.fit`, the expression

```
if(any(params < 0))
```

will be evaluated. If the conditional statement evaluates to `TRUE` then a penalty of `alpha` will be returned before the evaluation of the negative log-likelihood. The value of `alpha` is an argument to `ptproc.fit` and has a default value of zero. If restricting the parameters to be positive guarantees the positivity of the conditional intensity, then the above code would be an example of implementing the penalty function in (2).

The entire “`ptproc`” object can be accessed when inserting penalties into the log-likelihood. It can be accessed using the object name `ppobj`. The vector of model parameters can be access separately via the name `params`. The situation may arise when several statements must be inserted before a conditional statement can be evaluated. The `code` argument to `penalty` can be used for inserting several statements. There is an example of the usage of `code` in Section 5.1.

Finally, we can fit the model with

```
> fit <- ptproc.fit(ppm, optim.control = list(trace = 2), method = "BFGS", alpha = 1e+9)
initial value -341.202301
final value -360.516878
converged
```

In this case the computed MLE is 99.83, which is close to the true value of 100.

### 4.3 Constructing a Residual Process

The `ptproc` package provides a `residuals` method for a “`ptproc`” object which offers two possible procedures for generating the residual process. The user is given the option of generating ordinary thinned residuals or approximate thinned residuals. For ordinary thinned residuals, the user must provide a value  $m$  representing the minimum of the conditional intensity. For approximate thinned residuals, the user must specify a subsample size  $K$  to draw from the original points.  $K$  must be strictly smaller than the number of data points.

Continuing the example from Section 4.2, we can generate both kinds of residuals with

```
> r1 <- residuals(fit, type = "ordinary", m = params(fit))
> r2 <- residuals(fit, type = "approx", K = 20)
```

In this example, since the conditional intensity is constant the minimum is equal to the value of the estimated parameter. Therefore, we could use `m = params(fit)` when generating ordinary thinned residuals in the call to `residuals`.

The `residuals` method returns a matrix of points representing the thinned residual process. The number of rows in the matrix equals the number of points in the thinned process and there is one column for each dimension in the original dataset. If approximate residuals are used, the number of rows is always equal to  $K$ . These points can then be passed to other functions which test for uniformity.



## 4.4 Simulating a Point Process

A point process can be simulated using the random thinning methods of Lewis and Shedler (1979) and Ogata (1981), given a form for the conditional intensity. These procedures are implemented in the `ptproc.sim` function. This method of simulating a point process is very similar to generating a thinned residual process. Here, the user must specify a value  $M$  such that

$$\sup_{(t,\mathbf{x}) \in S} \lambda(t, \mathbf{x}) \leq M < \infty.$$

Then a homogeneous point process of rate  $M$  is generated in the domain  $S$ . Suppose there are  $L$  points in this realization. Then for  $i = 1, \dots, L$ , each point  $(t_i, \mathbf{x}_i)$  is deleted with probability  $1 - \lambda(t_i, \mathbf{x}_i)/M$ . The undeleted points form a realization of a point process with conditional intensity  $\lambda$ . Note that for fitted models `ptproc.sim` takes the domain for simulation to be the same domain in which the observed point process lies.

## 5 Examples

The various usages of the package are perhaps best demonstrated through examples. In this section we give an example of fitting a space-time linear model and a one-dimensional Hawkes-type cluster model.

### 5.1 Fitting a Simple Linear Model to Data

The dataset we use here consists of the times and locations of wildfires in the northern region of Los Angeles County, California. The spatial locations of the 313 wildfires occurring between 1976 and 2000 are shown in Figure 1.

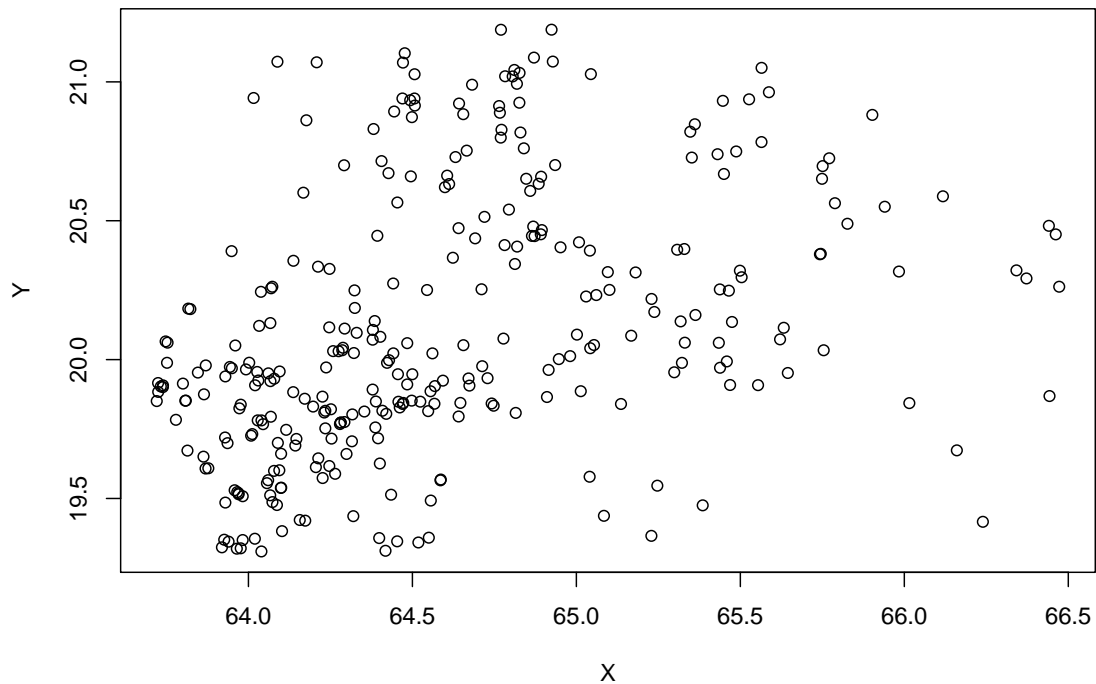


Figure 1: Northern Los Angeles County Wildfires (1976–2000). The direction north is towards the top of the figure.

The dataset is included in the package and can be accessed with the R function `data`. After loading the data one should find a  $313 \times 3$  matrix called `fires` in the global workspace where the first column contains the times of the wildfires and the second and third columns contain the  $x$  and  $y$  coordinates, respectively. The units of time in this dataset were produced by the `date` package and represent the number of days since January 1, 1960.

The model used here is a simple linear model with one parameter for each dimension and a background parameter. It is of the form

$$\lambda(t, x, y) = \mu + \beta_1 t + \beta_2 x + \beta_3 y. \quad (5)$$

We first load the data and construct a “`ptproc`” object.

```
> data(fires)
> ppm <- ptproc(fires, cond.int = linear.cond.int,
+             params = c(mu = .004, beta1 = 0, beta2 = 0, beta3 = 0))
```

The code for the function `linear.cond.int` is shown in Appendix A.

Clearly, it is possible for this conditional intensity function to take negative values so we will have to restrict the parameters somehow. One way would be to restrict all the parameters to be positive. However, that would likely restrict the model too much by not allowing any negative trend. In this case, since the conditional intensity is a plane, we can guarantee positivity by testing conditional intensity values at the 8 “corners” of the domain. If the conditional intensity is positive at those corners, then it must be positive in the middle points. The following statement constructs code to do this evaluation at the corners of the domain:

```
> extra.code <- paste("ranges <- as.list(as.data.frame(ppobj$ranges))",
+                    "corners <- expand.grid(ranges)",
+                    "ci <- evalCIF(ppobj, xpts = corners)", sep = ";")
```

Each argument to `paste` here is a single R expression. The expressions are concatenated together with the ‘;’ as a separator. After the conditional intensity has been evaluated at the 8 corners, we must test to see if any are negative. We will use the `penalty` and `condition` functions to modify the `condition` element of the “`ptproc`” object.

```
> condition(ppm) <- penalty(code = extra.code, condition = "any(ci < 0)")
```

The model can be fit without having to worry about negative values of the conditional intensity. We use the Nelder-Mead optimization method with the default 500 iterations and a tracing level of 2. Furthermore, we set the penalty parameter `alpha` equal to  $10^5$ .

```
> fit <- ptproc.fit(ppm, optim.control = list(trace=2), alpha = 1e+5)
```

After the parameters have been estimated, we can print the fitted model

```
> fit
Model type: LINEAR

Parameter Values:
      mu      beta1      beta2      beta3
1.610e-01 -8.986e-07 -2.180e-03 -7.746e-05

Initial Values:
      mu beta1 beta2 beta3
0.004 0.000 0.000 0.000

Fixed Parameters:
```

```
mu beta1 beta2 beta3
NA NA NA NA
```

```
Condition: expression(ranges <- as.list(as.data.frame(ppobj$ranges)), ...
```

and check the AIC

```
> AIC(fit)
[1] 3694.754
```

Often, the homogeneous Poisson model is a useful null model against which to compare more complex models. If the more complex model is truly capturing a feature of the data, its AIC value should be much lower than that of the homogeneous Poisson model. A fitted model's AIC can be compared against the homogeneous Poisson model by using the `summary` method:

```
> summary(fit)
Model type: LINEAR

Parameter Values:
      mu      beta1      beta2      beta3
1.610e-01 -8.986e-07 -2.180e-03 -7.746e-05
```

```
Model AIC:      3694.754
H. Pois. AIC:   3765.205
```

Here, we see that our simple linear model is in fact doing better but not very much so.

We can examine the fit of the model further by doing some residual analysis. We first try generating the ordinary thinned residuals. Since we need to know the minimum of the conditional intensity, the first two lines of code below compute the conditional intensity at the corners of the domain to find the minimum:

```
> corners <- expand.grid(as.list(as.data.frame(fit[["ranges"]]))
> ci.corners <- evalCIF(fit, xpts = corners)
> set.seed(100) ## Set seed so results can be reproduced
> r1 <- residuals(fit, "ordinary", m = min(ci.corners))
> pairs(r1)
```

The `pairs` plot of the residual process is shown in Figure 2. Alternatively, we could generate approximate thinned residuals. Here we will set the subsample size  $K$  equal to the number of points we obtained in the ordinary thinned residual process.

```
> set.seed(500)
> r2 <- residuals(fit, "approx", K = nrow(r1))
> pairs(r2)
```

The approximate thinned residuals are shown in Figure 3. One can see from both figures that neither residual process appears to be homogeneous Poisson. Both the ordinary and approximate residuals have a clear trend from the southwest corner to the northeast corner (see the bottom middle panel) and exhibit some clustering.

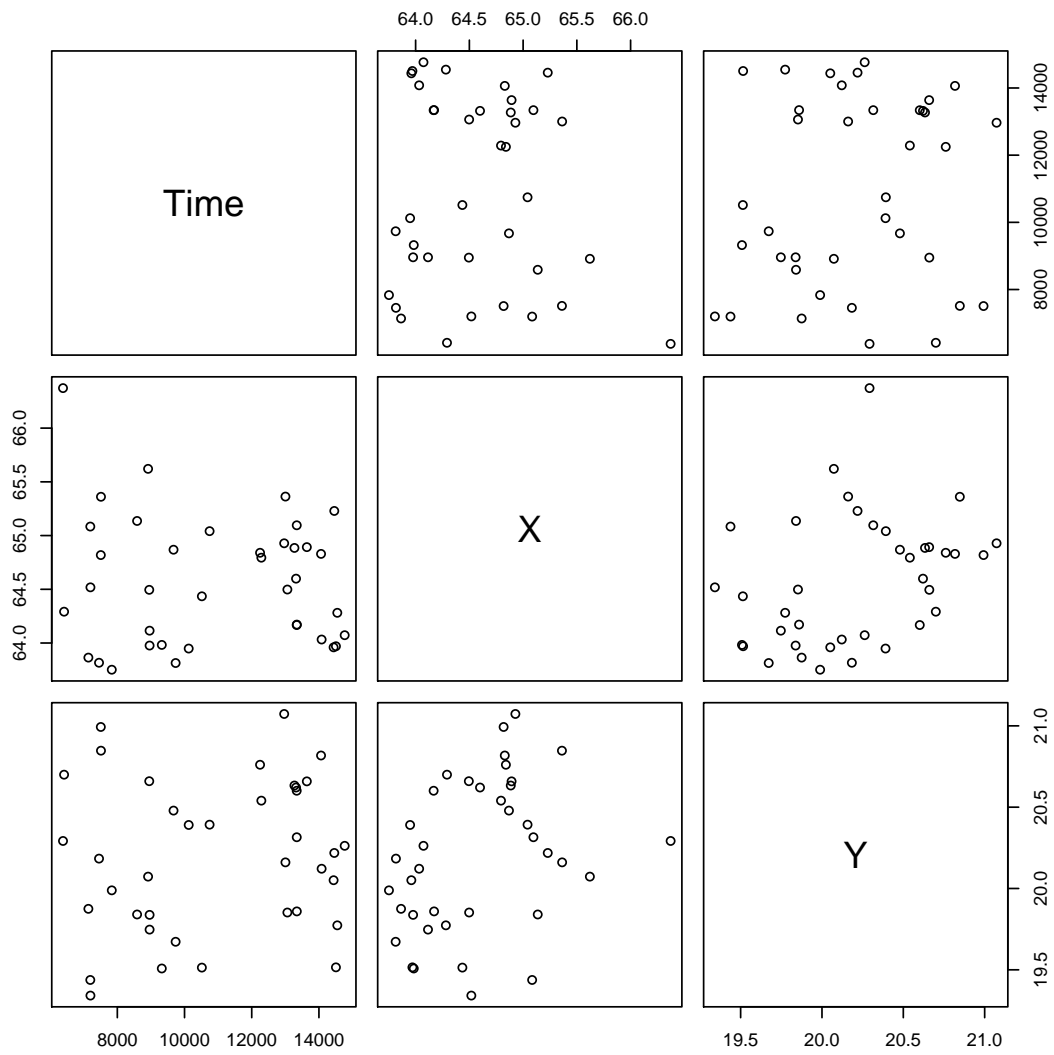


Figure 2: Ordinary thinned residuals from the fitted model in (5)

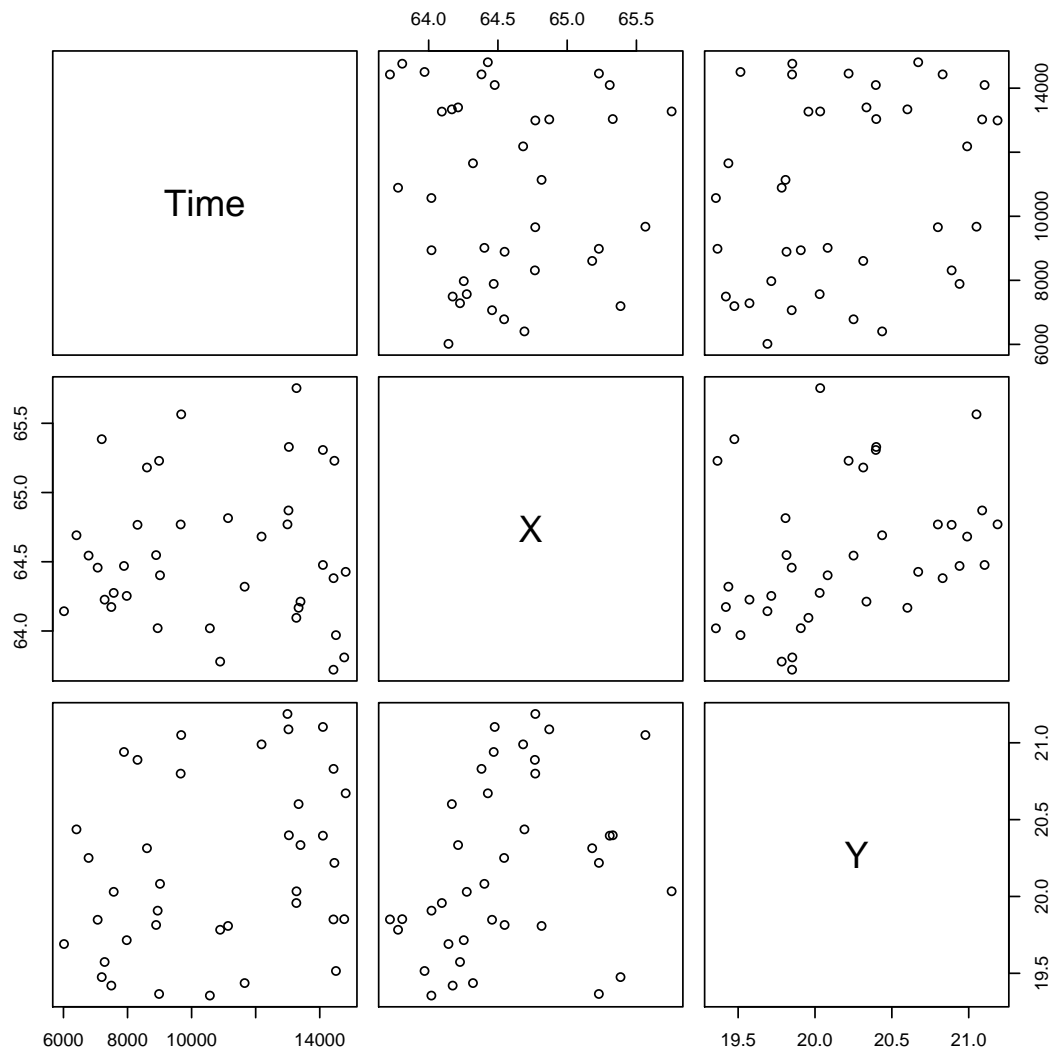


Figure 3: Approximate thinned residuals ( $K = 38$ ).

At this point we may wish to conduct various statistical tests on either the ordinary or approximate thinned residuals. We will use Ripley's  $K$  function to test for spatial clustering and inhibition. The  $K$  function measures, for a given  $h > 0$ , the expected number of points within distance  $h$  of a point in the process. The `splancs` package of Rowlingson and Diggle (1993) contains an implementation of the  $K$  function and has many other useful tools for analyzing spatial point patterns. The package is also available from CRAN. We will use the `khat` function to compute the  $K$  function and the `Kenv.csr` function to simulate confidence envelopes. For this example, we will use the ordinary thinned residual process for testing purpose since it appears to have a reasonable number of points in it.

```
> b <- make.box(fit, 2:3)
> h <- seq(.1, 2, .2)
> K <- khat(r1[,2:3], b, h)
> env <- Kenv.csr(nrow(r1), b, 2000, h)
```

Instead of plotting the raw  $K$  function we plot a standardized version

$$\hat{L}(h) = \sqrt{\frac{\hat{K}(h)}{\pi}} - h$$

where  $\hat{K}(h)$  is the estimated  $K$  function for distance  $h$ .

Figure 4(a-b) shows the standardized  $K$  function for the original data and the ordinary thinned residuals. From Figure 4(a) it is clear that the original data are clustered. The dotted red lines are confidence envelopes for the  $K$  function produced by 2000 simulations of a homogeneous Poisson process. The standardized  $K$  function in Figure 4(b) appears quite close to that of a homogeneous Poisson process, but the residual process still appears to be clustered. This would suggest that the model is not adequately accounting for some features in the data.

Because of the randomness involved in producing both sets of residuals, a second realization of the residual process would likely produce a different estimate of the  $K$  function. However, one strategy to deal with this could be to produce many realizations of the residual process, in turn producing many estimates of the  $K$  function. The the range of the  $K$  function for the various residual processes can be compared to the range of the  $K$  function for many simulations of a homogeneous Poisson process. If the ranges overlapped closely, then that would indicate the residual process may be a close to a homogeneous Poisson process.

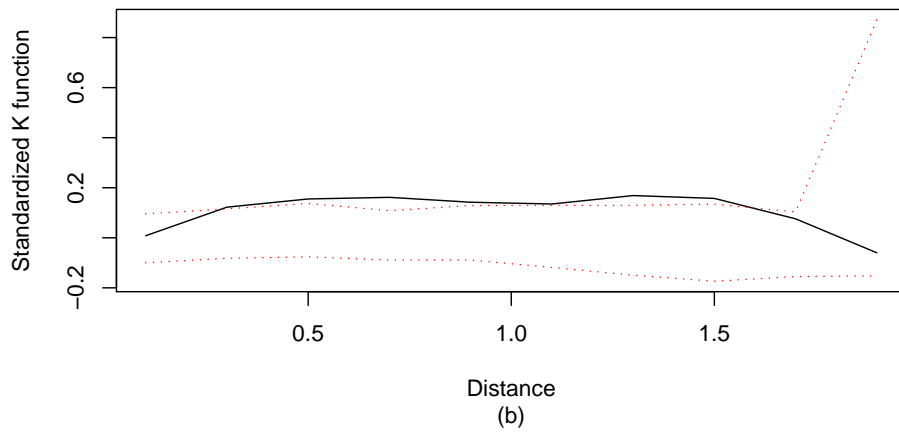
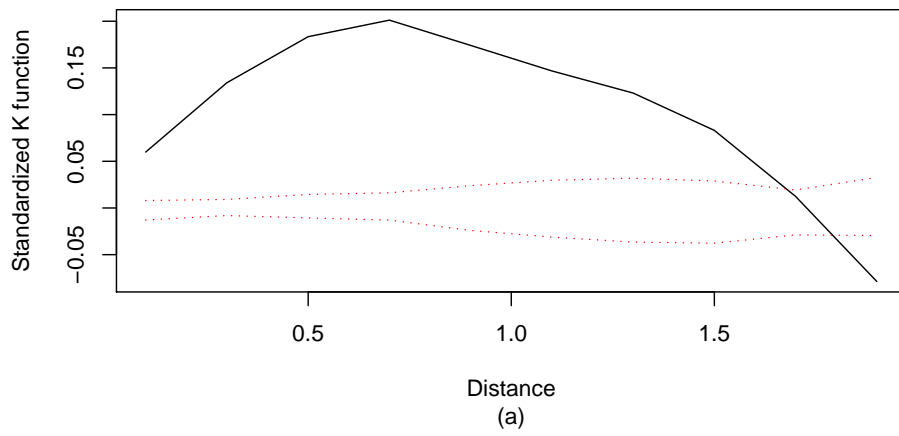


Figure 4: Standardized  $K$  function for (a) original data and (b) residual process.



## 5.2 Fitting a One-Dimensional Cluster Model

In this section we fit a Hawkes-type cluster model to the times of the wildfire occurrences dataset used in Section 5.1. In this example we show two useful diagnostic plots which can be produced from the one-dimensional residual process. We also show how one might fit a sequence of models when the model contains a variable number of parameters.

The conditional intensity model we use is

$$\begin{aligned}\lambda(t) &= \mu + \int_{-\infty}^t g(t-s) N(ds) \\ &= \mu + \sum_{t_i < t} g(t-t_i)\end{aligned}$$

where  $g$  is the trigger function

$$g(z) = \sum_{k=1}^K a_k z^{k-1} e^{-cz}.$$

Here the parameter  $\mu$  represents the background rate of occurrence while the parameters  $a_1, \dots, a_K$  and  $c$  control the level of clustering.  $K$  represents the order of the trigger function and selecting its value is an issue we discuss below. The code for this model can also be found in Appendix A.

We first construct the “`ptproc`” object (setting  $K = 2$ ) and then construct a penalty to ensure that the conditional intensity is positive. Here we simply restrict all of the parameters to be positive. We then fit the model with `ptproc.fit`.

```
> data(fires)
> times <- fires[,1]
> ppm <- ptproc(pts = times, cond.int = hawkes.cond.int,
+             params = c(mu = .004, C = .004, a = rep(0,2)))
> condition(ppm) <- penalty(code = NULL, condition = "any(params < 0)")
> fit <- ptproc.fit(ppm, optim.control = list(trace = 2), alpha = 1e+5, hessian = TRUE)
> summary(fit)
Model type: HAWKES
```

Parameter Values:

mu	C	a1	a2
8.372e-03	4.654e-02	4.011e-02	1.998e-08

Model AIC: 2440.086

H. Pois. AIC: 2730.803

The AIC values from the `summary` output show that the model does fit better than a homogeneous Poisson model. The estimated conditional intensity is shown with the data in Figure 5. The plot was constructed by using the package function `evalCIF` on a grid:

```
> x <- seq(5863, 14866, len = 2000)
> e <- evalCIF(fit, xpts = x)
> plot(x, e, type="l", xlab = "Times", ylab="Cond. Int.")
> rug(fit$pts, tick = 0.02)
```

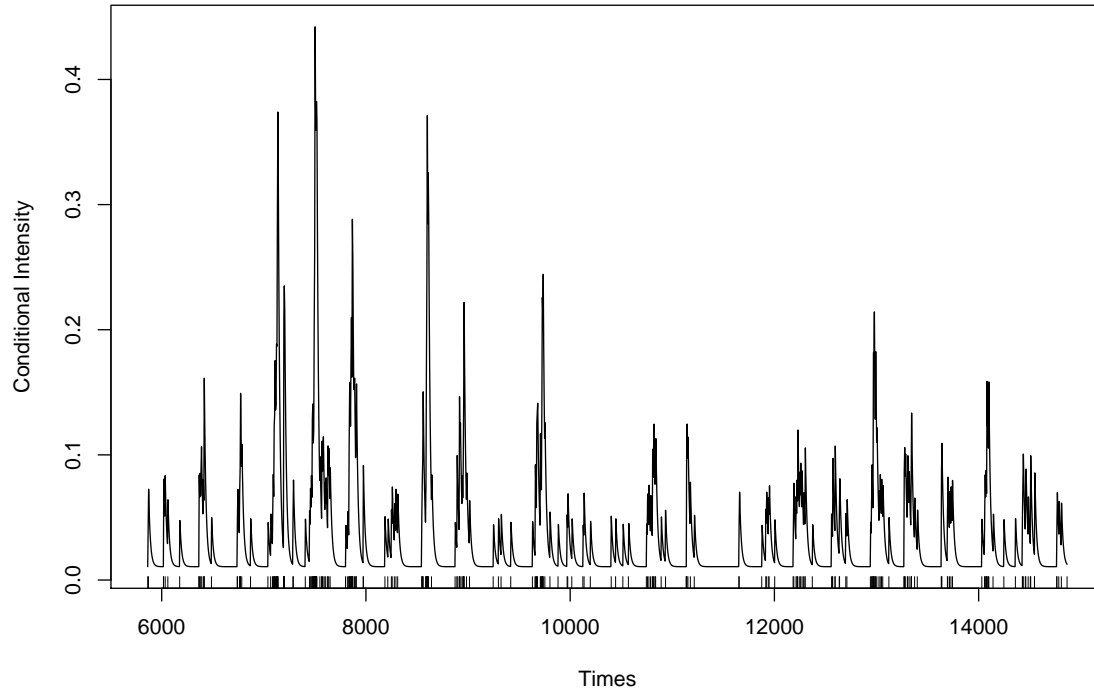


Figure 5: Estimated conditional intensity function with a rug plot of the wildfire occurrence times.

In the call to `ptproc.fit` we set the argument `hessian = TRUE`, which directs the `optim` function to estimate the Hessian matrix around the MLE. We can compute approximate standard errors from the diagonal of the inverse Hessian matrix. The standard errors for the parameters in this model are

mu	C	a1	a2
0.0013006	0.0051107	0.0055319	0.0000064

In this case  $\mu$ ,  $c$ , and  $a_1$  appear significantly different from zero, in that they are much larger than their standard errors. However, the  $a_2$  coefficient has a large standard error and is likely equal to zero.

A general issue with this kind of cluster model is the choice of  $K$ , the order of the polynomial in the trigger function. In the above example, we arbitrarily chose  $K = 2$ . However, we can use the AIC to compare a number of models with different values of  $K$ . In the following example, we show how this can be done, using values of  $K$  from 1 to 4.

```
> models <- vector("list", length = 4)
> for(k in 1:4) {
+   ppm <- ptproc(pts = fires[,1], cond.int = hawkes.cond.int,
+                 params = c(mu = .004, C = .004, a = rep(0,k)))
```

```

+   condition(ppm) <- penalty(code = NULL, condition = "any(params < 0)")
+   fit <- ptproc.fit(ppm, optim.control=list(trace = 2), alpha = 1e+5)
+   models[[k]] <- fit
+ }
> aic <- sapply(models, AIC)
> names(aic) <- 1:4
> aic
      1      2      3      4
2428.260 2440.086 2692.615 2746.713

```

It would appear that  $K = 1$  is the minimum AIC model of the four. In fact, for  $K = 4$  the model is doing worse than the homogeneous Poisson model (which, as we saw from the `summary` output, has an AIC of 2730.803).

We can further examine the goodness-of-fit of the  $K = 1$  model via residual analysis.

```

> fit <- models[[1]]
> set.seed(900)
> r <- residuals(fit, "ordinary", m = params(fit)[1])

```

There are a number of diagnostics one can use to assess the fit of a 1-dimensional residual process. One example is a log-survivor plot of the interevent times. If the model fits well, then the residual process should be homogeneous with rate  $m$  (where  $m$  is defined in (4) and specified in the call to `residuals`) and the interevent times should appear as i.i.d. exponential with mean  $1/m$ . The log-survivor plot of the interevent times of the residual process can be constructed with the `log.surv` function (included in the package) and is shown in Figure 6(a).

One can also check the stationarity of the residual process with the `stationarity` function. This function divides the domain into bins of a given (user-specified) length and counts the number of points falling into each bin is counted. The number of points in each bin is standardized by the theoretical mean and standard deviation and the standardized counts are plotted against the left endpoints of the bins. This plot is shown in Figure 6(b). The process generally stays within the bounds of a homogeneous Poisson process, but in one instance the count jumps beyond three standard deviations of the expected count. This jump could indicate a region of non-stationarity in the residual process and a feature of the data which is not taken into account by the model.

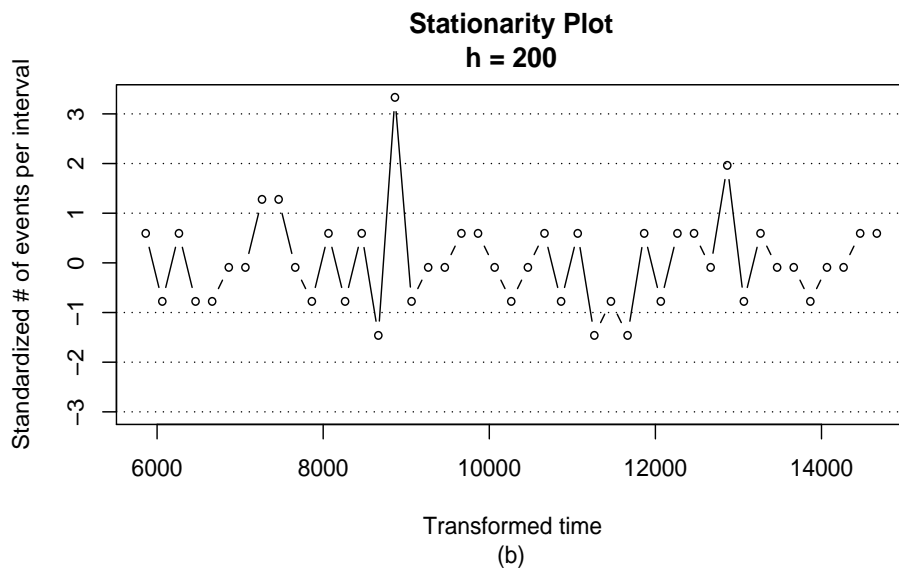
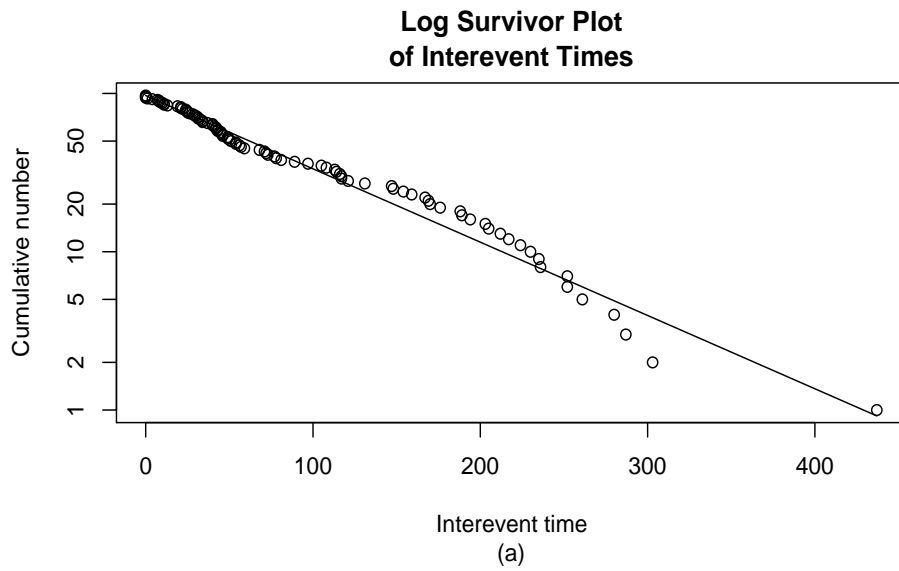


Figure 6: Diagnostics for 1-dimensional cluster model.

## 6 Future Work

In the future we hope to develop more functions for conducting model evaluation of multi-dimensional models. We would like to add more diagnostic plots and tools to aid in residual analysis. Also, simulation based methods of model evaluation can be very useful and have not been discussed here. We hope to increase the number of simulation based tools in future releases of the package.

## 7 Bug Reports

Please send any bug reports or suggestions to `rpeng@stat.ucla.edu`.

## 8 Acknowledgments

The author thanks Rick Paik Schoenberg, Frauke Kreuter, and Jane Chung for useful comments on the manuscript and many interesting discussions, and James Woods for providing the dataset used in the examples.

## References

- Belisle, C. J. P. (1992), “Convergence theorems for a class of simulated annealing algorithms on  $\mathbb{R}^d$ .” *Journal of Applied Probability*, 29, 885–895.
- Berman, M. and Turner, T. R. (1992), “Approximating point process likelihoods with GLIM,” *Applied Statistics*, 41, 31–38.
- Daley, D. J. and Vere-Jones, D. (1988), *An Introduction to the Theory of Point Processes*, Springer, NY.
- Diggle, P. J. (1983), *Statistical Analysis of Spatial Point Patterns*, Academic Press, NY, London.
- Gentleman, R. and Ihaka, R. (2000), “Lexical scope and statistical computing,” *Journal of Computational and Graphical Statistics*, 9, 491–508.
- Harte, D. (1998), “Documentation for the Statistical Seismology Library,” Tech. Rep. 98-10, School of Mathematical and Computing Sciences, Victoria University of Wellington.
- Ihaka, R. and Gentleman, R. (1996), “R: A language for data analysis and graphics,” *Journal of Computational and Graphical Statistics*, 5, 299–314.
- Lewis, P. A. W. and Shedler, G. S. (1979), “Simulation of nonhomogeneous Poisson processes by thinning,” *Naval Research Logistics Quarterly*, 26, 403–413.
- Nelder, J. A. and Mead, R. (1965), “A simplex algorithm for function minimization,” *Computer Journal*, 7, 308–313.
- Nocedal, J. and Wright, S. J. (1999), *Numerical Optimization*, Springer.
- Ogata, Y. (1978), “The asymptotic behavior of maximum likelihood estimators for stationary point processes,” *Annals of the Institute of Statistical Mathematics*, 30, 243–261.

- (1981), “On Lewis’ simulation method for point processes,” *IEEE Transactions on Information Theory*, 27, 23–31.
- (1983), “Likelihood analysis of point processes and its applications to seismological data,” *Bull. Int. Statist. Inst.*, 50, 943–961.
- (1988), “Statistical models for earthquake occurrences and residual analysis for point processes,” *Journal of the American Statistical Association*, 83, 9–27.
- (1998), “Space-time point process models for earthquake occurrences,” *Annals of the Institute of Statistical Mathematics*, 50, 379–402.
- (1999), “Seismicity analysis through point-process modeling: a review,” *Pure and Applied Geophysics*, 155, 471–507.
- Ogata, Y. and Tanemura, M. (1984), “Likelihood analysis of spatial point patterns,” *Journal of the Royal Statistical Society, Series B*, 46, 496–518.
- Rathbun, S. L. (1996), “Asymptotic properties of the maximum likelihood estimator for spatio-temporal point processes,” *Journal of Statistical Planning and Inference*, 51, 55–74.
- Rathbun, S. L. and Cressie, N. (1994a), “Asymptotic properties of estimators for the parameters of spatial inhomogeneous Poisson point processes,” *Advances in Applied Probability*, 26, 122–154.
- (1994b), “A space-time survival point process for a longleaf pine forest in southern Georgia,” *Journal of the American Statistical Association*, 89, 1164–1174.
- Ripley, B. (1976), “The second-order analysis of stationary point processes,” *Journal of Applied Probability*, 13, 255–266.
- (1979), “Tests of ‘randomness’ for spatial point patterns,” *Journal of the Royal Statistical Society, Series B*, 41, 368–374.
- Rowlingson, B. and Diggle, P. (1993), “Splancs: spatial point pattern analysis code in S-Plus,” *Computers and Geosciences*, 19, 627–655.
- Schoenberg, F. (1999), “Transforming spatial point processes into Poisson processes,” *Stochastic Processes and their Applications*, 81(2), 155–164.
- Schoenberg, F. P. (2002), “Multi-dimensional residual analysis of point process models for earthquake occurrences,” *Submitted*.
- Vere-Jones, D. (1970), “Stochastic models for earthquake occurrence,” *Journal of the Royal Statistical Society, Series B*, 32, 1–62.
- Vere-Jones, D. and Ozaki, T. (1982), “Some examples of statistical estimation applied to earthquake data,” *Annals of the Institute of Statistical Mathematics*, 34, 189–207.

## A Appendix: Code

### A.1 Simple Linear Model

Below is the code for the simple linear model used in Section 5.1. The conditional intensity is of the form

$$\lambda(t, x, y) = \mu + \beta_1 t + \beta_2 x + \beta_3 y.$$

```
linear.cond.int <- function(params, eval.pts, pts = NA, data = NULL, TT = NULL) {
  mu <- params[1]
  beta <- params[-1]

  if(is.null(TT)) {
    ## Evaluate
    ci <- mu + eval.pts %*% beta
    ci <- as.vector(ci)
  }
  else {
    ## Integrate
    total.vol <- prod(apply(TT, 2, diff))
    m.vol <- sapply(1:ncol(TT), function(i)
      {
        z <- TT[, -i, drop=FALSE]
        prod(apply(z, 2, diff))
      })
    d <- apply(TT^2 / 2, 2, diff)
    ci <- mu * total.vol + (beta * d) %*% m.vol
  }
  ci
}
```

### A.2 Hawkes-type Cluster Model

The version of Hawkes' self-exciting model used in Section 5.2 is

$$\lambda(t) = \mu + \sum_{t_i < t} \sum_{k=1}^K a_k (t - t_i)^{k-1} e^{-c(t-t_i)}.$$

```
hawkes.cond.int <- function(params, eval.pts, pts = NA, data = NULL, TT = NULL) {
  mu <- params[1]
  C <- params[2]
  ak <- params[-(1:2)]
  K <- length(ak)

  if(K < 1)
    stop("K must be >= 1")
  if(is.null(TT)) {
```

```

S <- sapply(as.vector(eval.pts), function(x, times, ak, C)
  {
    use <- times < x
    if(!is.na(use) && any(use)) {
      d <- x - times[use]
      k <- 0:(length(ak)-1)
      lxx <- outer(log(d), k) + (-C * d)
      sum(exp(lxx) %*% ak)
    }
    else 0
  }, times = as.vector(pts), ak = ak, C = C)
ci <- mu + S
}
else {
  Rfunc <- function(x, L, c) {
    k <- 0:(L-1)
    g <- gamma(k+1) / c^(k+1)
    o <- outer(x, k+1, pgamma, scale = 1/c)
    r <- t(t(o) * g)
    r
  }
  times <- as.vector(pts)
  ci <- mu*(TT[2,1]-TT[1,1])
  S <- double(2)

  for(i in 1:2) {
    use <- times < TT[i,1]

    if(any(use)) {
      r <- Rfunc(TT[i,1] - times[use], K, C)
      S[i] <- sum(r %*% ak)
    }
  }
  ci <- ci + (S[2] - S[1])
}
ci
}

```