BASS: Bayesian Analyzer of Event Sequences

E.Arjas[‡], *H.Mannila*[†], *M.Salmenkivi*[†], *R.Suramo*[†], *and H.Toivonen*[†]

[†]*University of Helsinki*
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23), FIN-00014 Helsinki, FINLAND
e-mail: {mannila, salmenki, suramo, htoivone}@cs.helsinki.fi
[‡]*University of Oulu*
Department of Mathematical Sciences
Linnanmaa, FIN-90570 Oulu, FINLAND
e-mail: Elja.Arjas@oulu.fi

Abstract. We describe the BASS system, a Bayesian analyzer of event sequences. BASS uses Markov chain Monte Carlo methods, especially Metropolis-Hastings algorithm, for exploring posterior distributions. The system allows the user to specify an intensity model in a high-level definition language, and then runs the Metropolis-Hastings algorithm on it.

Keywords. Bayesian analysis, event sequences, Metropolis-Hastings algorithm

1 Introduction

Statistical data are often expressed in the form of a sequences of events in time. Such data arise in a variety of applied areas, for instance, in telecommunications, quality control, as well as in infectious disease epidemiology and other areas of biostatistics. Intensity functions (see, e.g., Cox et al., 1984, and Arjas, 1989) are nowadays extensively used as a methodical tool in applications of this kind, to describe the instantaneous risk of observing the event of interest over time.

Frequently, it would be useful to obtain the posterior distribution function of the parameters of the model. Analytical methods are not feasible for any but the simplest models, but simulation methods, such as Monte Carlo Markov chain (MCMC) methods, can be used to obtain samples from the posterior distribution. However, writing an MCMC simulation program for each intensity model from scratch is tedious and error-prone.

In this paper we introduce BASS (Bayesian Analyzer of Event Sequences), a system that allows the user to specify an intensity model in a high-level definition language and then automatically produces a sample from the posterior distribution of the model parameters. The sample, on the other hand, can be used to obtain numerical Monte Carlo approximations of integrals with respect to the posterior. The program uses the Metropolis-Hastings algorithm (Hastings, 1970, Smith et al., 1993).

Section 2 introduces some related work. We present our problem setting in

Section 3. In Section 4 we give two examples of the use of BASS, and then describe briefly its current status of development. The syntax of the model specification language is presented in the Appendix.

2 Related work

The BUGS system (Bayesian inference Using Gibbs Sampling) (Spiegelhalter et al., 1994) has been very influential in the design of BASS. BASS is built on the use of the Metropolis-Hastings algorithm, whereas BUGS relies entirely on a special case of it, namely Gibbs sampling (Smith et al., 1993). In Gibbs sampling the user can only use conjugate distributions as priors. On the contrary, in BASS there is no such restrictions.

We are aware of two other tools for MCMC simulation. EMSS, an Environment for Montecarlo Simulation Studies (Sanchez et al., 1992), is a package intended to assist the users of Monte Carlo simulations at different levels. Another study (Neal, 1995) uses ordered over-relaxation to speed up the convergence of MCMC simulation in the context of training neural networks; the associated software seems to provide some general tools for using MCMC. It is not clear whether this approach is suitable for intensity models as well.

3 Problem setting

Our setting is the following: Let U be a set of units, A a set of event types, and T a set of time points in the observation interval $[T_0, T_1]$. An event E is a triple (u, a, t), where $u \in U$ is a unit, $a \in A$ is an event type, and $t \in T$ is the time at which the event occurred. Here units are the objects to which the considered events happen, and event types form a predefined set of possible outcomes.

For example, if we study leukemia in mice, a mouse can be viewed as a unit, and there are three event types: diagnosed leukemia, death, and censoring. If we study alarms in a telecommunication network, there are several units, namely the network elements (e.g., telephone exchanges, transmission devices), and several event types, such as different error messages emitted by the elements for fault diagnosis. Similarly, in a study concerning recidivism of criminal behaviour, a unit would correspond to an individual, and event types to a classification of crimes.

As input BASS gets an intensity model and a sequence of events. The model contains information about units and event types, their covariates, and especially intensities of the event types in the form of intensity functions. Every event type $a \in A$ has (a constant or) a piecewise constant intensity function $f_a(t, x_1^u, \ldots, x_j^u)$. Here t is time and $x_1^u, \ldots, x_j^u$ are covariates of the considered unit u. The likelihood function of the model has a product form: $\prod_{u \in U} \prod_{a \in A} L_u^a$, where L_u^a is the likelihood function of event type a for unit u. In general, L_u^a is of the form:

$$\exp(-\int_{T_0}^{T_1} f_a(t, x_1^u, \ldots, x_j^u)dt) \cdot \prod_{j=1}^{k} f_a(t_j, x_1^u, \ldots, x_j^u)$$

where $T_0 \leq t_1 < t_2 < \ldots < t_k \leq T_1$ are the event times of type a for unit u.

BASS compiles the given intensity model to a C program. The program uses the Metropolis-Hastings algorithm to form a sample from the posterior distribution of the model parameters.

4 Two examples of model specifications

Example 1 This is a simple example of a BASS intensity model with a single unit and one event type. Suppose that we are interested in modeling how often a component fails, and suppose that we have data about the earlier failure times of this component. In the example, the unit *comp* is the component whose failure times we want to study. The event type is *failure*, and it has one attribute, *failure_time*. The intensity function *failure_rate* tells how often that component fails on average. The function is assumed to be piecewise constant with k pieces, where k, according to the prior, has a geometric distribution over the observation interval. The unknown values (levels) of the intensity function and its jump points are supposed to have gamma and uniform (on an interval) prior distributions, respectively.

To get the posterior distribution we need the prior probability distribution of each parameter in the model. Those distributions are given inside update functions (u-functions) of the model, for instance in u-function *failure_rate* in this example.

```
unit comp {}                                % unit definition, no attributes

event type failure                          % event type definition
{
   attributes failure_time;                 % one attribute: failure time
   intensity failure_rate(failure_time);    % intensity function declaration
   data in file failure.data (comp,         % name and format of the
                       failure_time);       % data file
}

u-function failure_rate(failure_time)       % update function definition
{
   piecewise constant:                      % piecewise constant intensity:
   prior pieces k ~ dgeom(0.1);             % priors of number of pieces,
   prior levels l ~ dgamma(2.0, 4.0);       % levels of the intensity function,
   prior jump_points                        % and the jump points of
       j ~ dunif (0.0, 10.0);               % the intensity function
}
```

Example 2 In this example we consider seismological data. We have a single unit, *area*, and two event types. Suppose we are interested in modeling the frequency of earthquakes in a certain area. As is well known, increased seismic activity is often triggered by a large main shock, and then continues in the form of several smaller aftershocks. This kind of dependence mechanism is described naturally in terms of a shot-noise model. The two event types are *main_shock* and *aftershock*. In the simplest version, the intensity of main shocks, *main*, can be assumed to have a constant value, *baseline*. The intensity of aftershocks, *after*, is here modeled in terms of a shot-noise effect; the value of this function goes up immediately after the main shock from value zero, reaching the level *high*, and staying there for a period of *high_time*.

```
constant high_time ~ dunif(0.0, 5.0);     % prior of this constant
constant amount ~ dunif(1.0, 80.0);       % prior of this constant

unit area {}                              % unit definition, no attributes

event type main_shock                     % event type definition
{
   attributes t;                          % one attribute: t (time)
   intensity main(t);                     % intensity function declaration
   data in file main.data (area, t);      % name and format of the
}                                         % data file

event type aftershock                     % event type definition
{
   attributes t;                          % one attribute: t (time)
   intensity after(t);                    % intensity function declaration
   data in file after.data (area,t);      % name and format of the
}                                         % data file

u-function main()                         % update function definition
{
   constant:                              % intensity is constant
   prior baseline ~ dunif(0.0, 10.0);     % prior of this constant
}

function after(t)                         % function definition
{                                         % contains a shot-noise effect
   if (t-last(main_shock) ≤ high_time)
   then amount
   else 0;
}
```

5 Status of BASS and future plans

BASS is currently under construction. At present, the system is able to handle simple models (e.g., shot-noise interactions) and moderate amounts of data. Some kind of optimization is needed when dealing with greater amounts of data or more complex models in the meaning of many parameters. The results are promising so far.

References

Arjas, E. (1989). "Survival models and martingale dynamics". Scandinavian Journal of Statistics, 16, p. 177-225.

Cox, D. R.; Oakes, D. (1984). *Analysis of Survival Data.* Monographs on Statistics and Applied Probability 21. Chapman & Hall.

Hastings, W. K. (1970). "Monte Carlo sampling methods using Markov chains and their applications". Biometrica 57, p. 97-109.

Neal, R. M. (1995). *Bayesian Learning for Neural Networks.* Ph.D. thesis, Department of Computer Science, University of Toronto.

Sanchez, A.; Ocaña, J.; Ruiz de Villa, C. (1992). "An Environment for Montecarlo Simulation Studies (EMSS)". Symposium of Computational Statistics (COMPSTAT'92), Vol. 2, p. 195-199.

Smith, A. F. M.; Roberts, G. O. (1993). "Bayesian Computation via the Gibbs Sampler and Related Markov Chain Monte Carlo Methods". Journal of the Royal Statistical Society, Series B, 55, No. 1, pp. 3-23.

Spiegelhalter, D.; Thomas, A.; Best, N.; Gilks, W. (1994). "BUGS* Bayesian inference Using Gibbs Sampling", Version 0.30. MRC Biostatistics Unit, Institute of Public Health. Cambridge, UK.

Appendix: Syntax of the model specifications

The current syntax of the model specification is presented using BNF (Backus Naur Format). Terms between the marks { and } can appear zero times or more. The mark | means alternativity, and marks [and] optionality. Words that are written in **boldface** belongs to the definition language of BASS.

BASS-definition :== [setting-part] unit-part event_type-part funct-part
setting-part :== {settings **;**} { constants **;**} | { constants **;**} { settings **;**}
settings :== **set** numerical_setting-term = integer
    | **set** non-numerical_setting-term = identifier
numerical_setting-term :==
    (**inititerations** | **iterations** | **t_end** | **change_pieces**)
non-numerical_setting-term :== **time**
constants :== **constant** identifier ~ distr

distr :== distr-name (distr-arguments)
distr-name :== (**dgeom** | **dgamma** | **dunif** | **dnorm** | **dpoisson**)
distr-arguments :== real {, real}
unit-part :== unit-name {[attribute-part ; file-name (arguments) ;] }
unit-name :== **unit** identifier
attribute-part :== **attributes** identifier {, identifier}
file-name :== **data in file** identifier
arguments :== identifier {, identifier}
event_type-part :== event_type-def {event_type-def}
event_type-def :== **event type** identifier { [attribute-part ;]
    intensity-funct ; file-name (arguments) ; }
intensity-funct :== **intensity** identifier ([arguments])
funct-part :==
    {(funct | u-funct)} u-funct {(funct | u-funct)}
u-funct :== **u-funct** identifier ([arguments]) {u-funct-body }
u-funct-body :== **constant: prior** identifier ~ distr ;
    | **piecewise constant: prior pieces** identifier ~ distr ;
    prior levels identifier ~ distr ;
    prior jump_points identifier ~ distr ;
funct :== **funct** identifier ([arguments]) { funct-body }
funct-body :== test-body | statement
test-body :== **if** (statement comparing_operator statement)
    then statement **else** statement
statement :== statement_term { arithmetic_operator statement }
    | (statement_term { arithmetic_operator statement })
statement_term :== real | identifier | funct-term
funct-term :== identifier ([arguments])
    | **last** (identifier)