lgcp Package Vignette: Spatiotemporal Prediction in Log-Gaussian Cox Processes

B. Taylor, T. Davies, B. Rowlingson, P.Diggle

April 20, 2012

Contents

1 Introduction **2**

2 Walk-through Example, Wales **2**

3 Reading In and Converting Data **4**

3.1 Simulation of Data 4

3.1.1 Simulation of Data with District-level `spatialAtRisk` 4

3.1.2 Simulation of Data with `spatialAtRisk` described by a Kernel Density Estimate of Population 5

3.1.3 More Complex Temporal-At-Risk Models 7

3.2 Analysis of a Dataset 7

3.3 Estimating the Spatial/Temporal Component 8

3.4 Estimating the Parameters 9

3.5 Performing Predictive Inference 11

3.6 Post-processing 12

3.6.1 Extracting Information 13

3.6.2 Plotting 13

3.6.3 MCMC diagnostics 13

3.6.4 NetCDF 14

3.6.5 Plotting Exceedance Probabilities 16

4 Advanced use of lgcp **18**

4.1 Writing Adaptive MCMC schemes 18

4.1.1 A 'Simple' Example: `constanth` 18

4.1.2 A More Complex Example: `andrieuthomsh` 19

A Spatio-temporal Log Gaussian Cox Processes **21**

B Inference **22**

B.1 Discretising and the Fast-Fourier Transform 23

B.2 The Metropolis-Adjusted Langevin Algorithm 23

C Adaptive MCMC **24**

D Monte Carlo Averaging and Exceedance Probabilities **25**

E Rotation **26**

F Gradient Truncation **27**

1 Introduction

This vignette provides an introduction to **lgcp** by means of a walk-through example; the theory necessary to understand the model is covered in the appendices and [17].

The **lgcp** package makes extensive use of **spatstat** functions and data structures [3]. Other important dependencies are: the **sp** library, which also supplies some data structures and functions [4]; the suite of covariance functions provided by the **RandomFields** package [16]; the **rpanel** package to facilitate minimum-contrast parameter estimation routines [5]; and the **ncdf** package for rapid access to massive datasets for post-processing [11].

2 Walk-through Example, Wales

Boundary files for Wales can be obtained from the Great Britain data files from the GADM (`http://www.gadm.org/country`) website. The following script will download the level 1 (England,

Scotland, Wales, Northern Ireland) and level 2 (counties and regions) data in .RData format, and put them in a temporary directory:

```
td = tempdir()
mapdata = c("GBR_adm1.RData","GBR_adm2.RData")
src = "http://www.gadm.org/data/rda/%s"

for(mapfile in mapdata){
  download.file(sprintf(src,mapfile),
                destfile=file.path(td,mapfile))
}
```

The first polygon needed is the observation window, which is the boundary of Cymru (in English, "Wales"), so it is extracted and converted with the following:

```
# load as object 'gadm'
load(file.path(td,"GBR_adm1.RData"))
# extract the Wales polygons
cymru_border <- gadm[gadm$NAME_1=="Wales",]
# convert to OS Grid reference
cymru_border <- spTransform(cymru_border,CRS("+init=epsg:27700"))
# supress some warnings
spatstat.options(checkpolygons = FALSE)
# convert to owin, and simplify
W <- as(cymru_border,"owin")
W <- simplify.owin(W,dmin=2000)
spatstat.options(checkpolygons = TRUE)
```

The transformation to OS Grid units is done so that the resulting locations have valid Euclidean distances, which is not the case with latitude-longitude coordinates.

Note that the `simplify.owin` step is fairly crucial in this example. Without simplifying the observation window, some of the later internal routines can run for a LONG time.

Next, the county-level data are loaded, Wales extracted, and population counts attached to the county polygons:

```
data(wpopdata) # obtained from ONS, population in thousands
load(file.path(td,"GBR_adm2.RData"))
cymru <- gadm[gadm$NAME_1=="Wales",]
cymru <- spTransform(cymru,CRS("+init=epsg:27700"))
idx <- match(wpopdata$ID,cymru$ID_2)
cymru$atrisk <- wpopdata$Mid.2010[idx]
```

3 Reading In and Converting Data

In the following example `x`, `y` and `t` are vector R objects giving the location and time of events, `tlim` is the time window, and `win` is a **spatstat** object of class `owin` specifying the polygonal observation window, in this case, a 10×10 square.

```
> data <- cbind(x,y,t)
> tlim <- c(0,100)
> win <- owin(poly=list(x=c(0,10,10,0),y=c(0,0,10,10)))
> win

window: polygonal boundary
enclosing rectangle: [0, 10] x [0, 10] units
```

For the purposes of parameter estimation and spatio-temporal prediction, the first task for the user is to convert this into a space-time planar point pattern object ie. one of class `stppp`, provided by **lgcp**. An object of class `stppp` is easily created:

```
> xyt <- stppp(list(data=data,tlim=tlim,window=win))
> xyt

Space-time point pattern
 planar point pattern: 1000 points
window: polygonal boundary
enclosing rectangle: [0, 10] x [0, 10] units
   Time Window : [ 0 , 100 ]
```

In the next section, it is discussed how to simulate data on the Wales observation window.

3.1 Simulation of Data

In this section it is demonstrated how to simulate data based on two different assumptions about the population at risk - first by using county-level populations and then by a kernel-density estimate from the town population counts.

In both cases, it will be assumed that the temporal trend $\mu(t)$ (see appendix for details) is constant in time, with the expected number of cases per day being 100.

3.1.1 Simulation of Data with District-level `spatialAtRisk`

To run the simulation a time-range (`tlim`) and parameter values (`sigma`, `phi` and `theta`) are required. Also, the simulation routine needs to know the quantities $\lambda(s)$ and $\mu(t)$ (see the appendices for further details of the parameters and these two functions).

In this example the `spatialAtRisk` object `sar1` (specifying $\lambda(s)$) describes the case where inhomogeneity in the at-risk population is aggregated to district level.

The grid size used for simulation is printed, this is the size on which the simulated data was generated; the fast Fourier transform (FFT, used in computation – see appendices) takes place on a grid four times the area of the printed grid. A progress bar is displayed during simulation and a warning message issued. The warning message is issued because the user-defined `mut` is not 'scaled', thus the literal interpretation of $\mu(t)$ is the expected number of cases in the unit time interval containing t. The warning can be disabled, see `?constantInTime` for example.

```
tlim <- c(1,100)
sigma <- 4
phi <- 7000 # distance in metres
theta <- 0.3

sar1 <- spatialAtRisk(cymru)
mut <- constantInTime(100,tlim=c(1,100))

xyt1 <- lgcpSim(owin=W,
        tlim=tlim,
        spatial.intensity=sar1,
        temporal.intensity=mut, # constant in time
        cellwidth = 3000, # gives a 128x128 simulation grid
        model.parameters=lgcppars(sigma=sigma,phi=phi,theta=theta),
        plot=FALSE)
```

To plot the `spatialAtRisk` object, use the following:

```
spplot(sar1,"atrisk")
```

This produces Figure 1 (note that the colour scheme on this plot can be user-defined via the `sp.theme` command). A corresponding plot of the data can be produced with

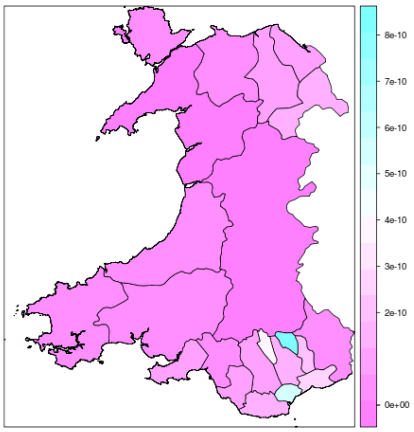
```
plot(xyt1,axes=TRUE,main="xyt1")
```

3.1.2 Simulation of Data with `spatialAtRisk` described by a Kernel Density Estimate of Population

In this second example, a continuous $\lambda(s)$ specifies the at-risk population. This is estimated from a kernel density estimate of population data. To load the town population data, type:

```
data(wtowns) # details of 199 welsh towns
data(wtowncoords) # coordinates of the 199 welsh towns
```

Figure 1: . Plot of `sar1` – population density Funiform over districts.



To get the kernel density estimate of the population, convert to a `ppp` object and then use the `density` function with the population as weights:

```
cymruppp <- ppp(wtowncoords[,1],wtowncoords[,2],
  marks=as.data.frame(wtowns[,2:3]),window=W)
den <- density(cymruppp,sigma=10000,weights=cymruppp$marks$Population.2001.)
```

The object `den` can now be converted into a `spatialAtRisk` object, and the simulation can be run:

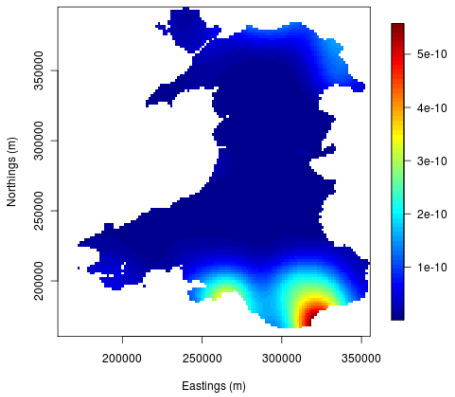
```
sar2 <- spatialAtRisk(den)

xyt2 <- lgcpSim(owin=W,
        tlim=tlim,
        spatial.intensity=sar2,
        temporal.intensity=mut, # constant in time
        cellwidth = 3000, # gives a 128x128 simulation grid
        model.parameters=lgcppars(sigma=sigma,phi=phi,theta=theta),
        plot=FALSE)
```

To plot the `spatialAtRisk` object, use the following:

```
image.plot(sar2$X,sar2$Y,sar2$Zm,xlab="Eastings (m)",
                  ylab="Northings (m)",axes=TRUE)
```

This produces Figure 2.

Figure 2: . Plot of `sar2` – population density described by a kernel density estimate.



3.1.3 More Complex Temporal-At-Risk Models

In the previous simulations the temporal trend was set to a constant 100 cases per day. In [7] a more complex model is used which incorporates a linear increasing trend, a seasonal pattern, and a day-of-the-week effect. This can be replicated here by passing an appropriate function to the `temporalAtRisk` function:

```
mutfun <- function(t){
    dotw <- c(1.76,1.82,1.76,1.78,2.12,2.24,1.92) # day of the week effect
    names(dotw) <- c("Tue","Wed","Thur","Fri","Sat","Sun","Mon")
    alpha1 <- -0.120
    alpha2 <- -0.013
    beta1 <- -0.083
    beta2 <- 0.054
    omega <- 2*pi/365
    gamma <- 0.00074
    return(exp(dotw[t%%7+1]+alpha1*cos(omega*t)+beta1*sin(omega*t)+
      alpha2*cos(2*omega*t)+beta2*sin(2*omega*t)+gamma*t))
}
mut <- temporalAtRisk(mutfun,tlim=c(1,365*2))
```

where the various constants come from fitting a GLM to the calibration data.

3.2 Analysis of a Dataset

In this section a walk-through analysis of the simulated data `xyt2` is given. The outline procedure is:

1. estimate the fixed spatial and temporal components ($\lambda(s)$ and $\mu(t)$);

2. estimate the pair correlation function (or equivalent);
3. using 2, compute estimates of the spatial parameters (σ and ϕ);
4. using 3, compute an estimate of the temporal parameter, θ;
5. produce predictions using `lgcpPredict`.

For the analysis of the data, the units of distance were converted to km instead of metres:

```
xyt2 <- rescale(xyt2,1000)
```

This is just to improve the look of axis-division labels.

3.3 Estimating the Spatial/Temporal Component

To estimate $\lambda(s)$ from the data, use for example,

```
lambdaest <- lambdaEst(xyt2)
```

This brings up an **rpanel** interface like the one shown in Figure 3. When the function is first called, an appropriate bandwidth is selected using the default method for `density.ppp`. Should the user wish to have this selected in a statistically principled manner, then simply clicking the 'OK' button will save the resulting density estimate. Now convert this kernel density estimate to a `spatialAtRisk` object:

```
sarest <- spatialAtRisk(lambdaest)
```

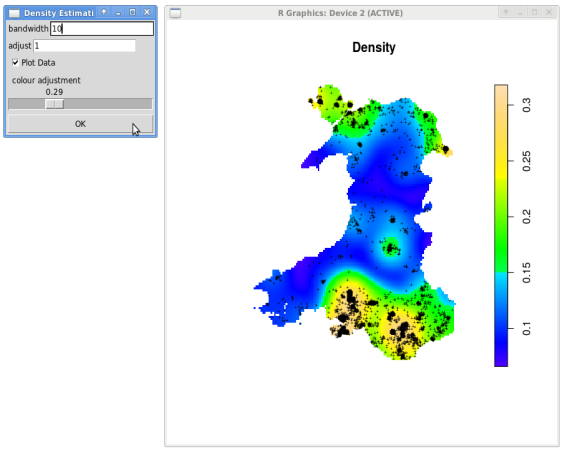
Alternatively, if the user is not happy with the density estimate selected with the default methods, then the bandwidth may be adjusted by eye. Since `image` plots of these kernel density estimates may not have appropriate colour scales, the ability to adjust this is given with the slider 'colour adjustment'. With colour adjustment set to 1, the default `image.plot` for the equivalent pixel image object is shown and for values less than 1, the colour scheme is more spread out, allowing the user to get a better feel for the density that is being fitted. NOTE: colour adjustment does not affect the returned density and the user should be aware that the returned density will 'look like' that displayed when colour adjustment is set equal to 1.

In fitting this model, a 'constant-in-time' $\mu(t)$ will be used, that is $\mu(t) = k$ for some k. To calibrate this k to the dataset at hand, use:

```
mutest <- constantInTime(xyt2)
```

An alternative, for a more complex $\mu(t)$ might be:

Figure 3: Choosing a kernel density estimate of $\lambda(s)$




```
temporalAtRisk(mutfun,xyt=xyt2,tlim=xyt2$tlim)
```

which would return the GLM fitted values function presented in Section 3.1.

3.4 Estimating the Parameters

Parameter estimation by the **lgcp** minimum contrast methods must be done in a certain order, as outlined at the start of the section.

To estimate the spatial correlation parameters σ and ϕ, either the pair correlation function or the inhomogeneous K function must first be obtained [2]. Following [6] and [7], these two functions are averaged over time instances. For the pair correlation function, the command is:

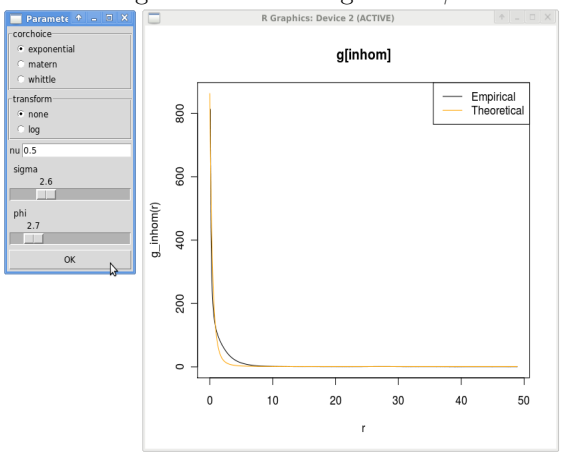
```
gin <- ginhomAverage(xyt2,spatial.intensity=sarest,temporal.intensity=mutest)
```

(for the inhomogeneous K function, the equivalent command is `KinhomAverage`). The parameters are then estimated using:

```
sigmaphi <- spatialparsEst(gin,sigma.range=c(0,10),phi.range=c(0,20))
sigma <- sigmaphi$sigma
phi <- sigmaphi$phi
```

which produces the interactive plot in Figure 4, the task is to match the orange theoretical function with the empirical equivalent as would normally be obtained from the functions `pcfinhom` for **spatstat** `ppp` objects (for example). The user can choose between the exponential, Matérn or Whittle correlation shapes, see `?CovarianceFct` from the **RandomFields** package; for the latter two forms, the additional parameter ν can be also be set by the user and is returned as `sigmaphi$nu` for example.

Figure 4: Estimating σ and ϕ

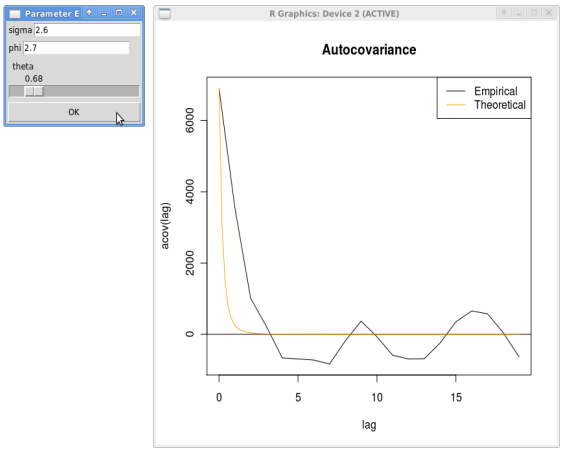


The temporal correlation parameter, θ, is estimated using the function `thetaEst`; this requires σ and ϕ to have been estimated first.

```
theta <- thetaEst(xyt2,spatial.intensity=lambdaest,temporal.intensity=mutest,
                  sigma=sigma,phi=phi,theta.range=c(0,5))
```

this brings up the interactive plot shown in Figure 5.

Figure 5: Estimating θ



Note that the parameter estimates obtained here were not particularly close to the values used to simulate the data: this is because the amount of data generated was relatively small and the time interval over which observations took place was relatively short. With more data and a longer observation window, the estimates of the parameters improve. Furthermore, minimum contrast methods are only approximate methods, one alternative would be a principled Bayesian approach, but this is computationally expensive.

3.5 Performing Predictive Inference

All of the necessary ingredients are now present in order to perform predictive inference, in this section prediction is considered for the time interval `T=50` using data from the previous 5 days (`laglength=5`). A MALA run of length 50,000 with a burn-in of 10,000, retaining every 40th iteration is requested and adaptive MCMC is used to tune the algorithm (see Appendix C). Also requested is a dump of the MCMC run to disk (set with `dump2dir`) and an online computation of 'exceedance probabilities' (see Appendix D) – the first line of code below sets up a function for exactly this purpose. Note that as data has been dumped to disk, the exceedance probabilities could have been computed after the run has finished (rather than online, which slows down the running time).

```
exceedfunc <- exceedProbs(c(1.5,2,3))

lg <- lgcpPredict(xyt=xyt2,
  T=50,
  laglength=5,
  model.parameters=lgcppars(sigma=sigma,phi=phi,theta=theta),
  cellwidth=6,
  spatial.intensity=lambdaest,
  temporal.intensity=mutest,
  mcmc.control=mcmcpars(mala.length=50000,burnin=10000,
  retain=40,MCMCdiag=5,
  adaptivescheme=andrieuthomsh(inith=1,alpha=0.5,C=1,
  targetacceptance=0.574)),
  output.control=setoutput(gridfunction=
  dump2dir(dirname="C:/My_MALA_runs/"),
  gridmeans=MonteCarloAverage("exceedfunc",lastonly=TRUE)))
```

This initially produces the following output, an explanation of which follows:

```
FFT Grid size: [128 , 128]
WARNING: disk space required for saving is approximately 187.61 Mb, continue?

1: yes
2: no

Selection: 1
Note: to bypass this menu, set forceSave=TRUE in dump2dir
Warning in lgcpPredict(xyt = xyt2, T = 50, laglength = 5, model.parameters = lgcppars(sigma = sigm
  Time 45: 5 data points lost due to discretisation.
Computing gradient truncation ...
Using gradient truncation of 722369
Netcdf file: C:/My_MALA_runs/simout.nc created
```

The size of the FFT grid used is displayed (though if it is computationally advantageous to rotate the observation window, a message is displayed, giving the user the option to set `autorotate=TRUE` in `lgcpPredict`, see Appendix E for further details on rotation). The FFT grid size of 128×128 gives an output grid size of 64×64. As a dump to disk of (a subset of) grids of this size was requested, the user is prompted in case the dump is too large, the option to bypass the message is also possible. A warning message is issued to inform the user that 5 observations from one of the time points have been dropped, this is due to the choice of spatial discretisation used: the coarser the grid the more likely it is for points near the edges to be excluded from the analysis. The user is then informed that the gradient truncation parameter has been computed (see Appendix F) and also a netCDF file for dumping the data.

Whilst the MCMC algorithm is running, a progress bar is displayed.

3.6 Post-processing

Having run the MALA, the stored output `lg` is an object of class `lgcpPredict`. Typing `lg` into the console prints out information about the run:

```
> lg

lgcpPredict object.

  General Information
  -------------------
      FFT Gridsize: [ 128 , 128 ]

              Data:
       Time |       45       46       47       48       49       50
     Counts |       57       15       23       56       37       87

        Parameters: sigma=2.6, phi=2.7, theta=0.68
    Dump Directory: C:/My_MALA_runs/

     Grid Averages:
         Function Output Class
       exceedfunc        array

        Time taken: 3.959 hours

  MCMC Information
  ----------------
    Number Iterations: 50000
              Burn-in: 10000
```

```
       Thinning: 40
Mean Acceptance: 0.587
Adaptive Scheme: andrieuthomsh
         Last h: 1.67430730310916e-05
```

Information returned includes the FFT grid size used in computation; the count data for each day; the parameters used; the directory the simulation was dumped to, if this was specified; a list of `MonteCarloAverage` functions together with the **R** class of their returned values; the time taken to do the simulation; and finally, information on the MCMC run.

3.6.1 Extracting Information

The cell-wise mean and variance of Y computed via Monte Carlo can always be extracted using `meanfield(lg)` and `varfield(lg)`, respectively. The calls `rr(lg)`, `serr(lg)` and `intens(lg)` return respectively the Monte Carlo mean relative risk (the mean of $\exp\{Y\}$), the standard error of the relative risk and the estimated cell-wise mean Poisson intensity. The x and y coordinates for the grid output are obtained via `xvals(lg)` and `yvals(lg)`. If invoked, the commands `gridfun(lg)` and `gridav(lg)` return respectively the `gridfunction` and `gridmeans` options of the `setoutput` argument of the `lgcpPredict` function, whilst `window(lg)` returns the observation window.

Note that the structure produced by `gav <- gridav(lg)` is a `list` of length 2. The first element of `gav`, retrieved with `gav$names`, is a list of the function names given in the call to `MonteCarloAverage`. The second element, `gav$output`, is a list of the function outputs; the ith element in the list being the output from the function corresponding to the ith element of `gav$names`. To return the output for a specific function, use the syntax `gridav(lg,fun="exceed")`, which in this case returns the exceedance probabilities, for example.

3.6.2 Plotting

A plot of the Monte Carlo mean relative risk ($\exp\{Y\}$) can be obtained with the command

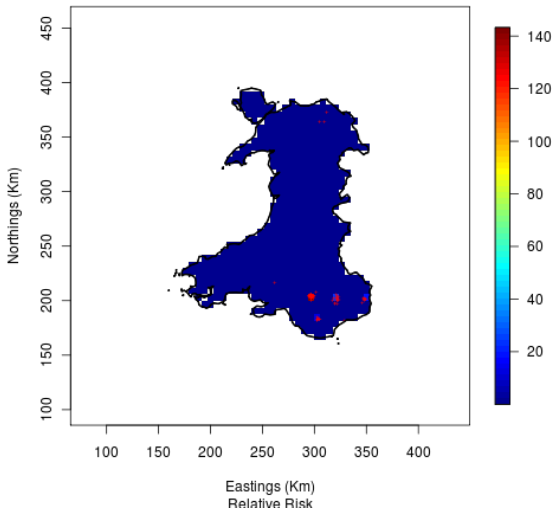
```
plot(lg,xlab="Eastings (km)",ylab="Northings (km)")
```

this produces a series of plots corresponding to each time step under consideration; the plot shown in Figure 6 is from the last time step, time 50. The cases for each time step are also plotted by default.

3.6.3 MCMC diagnostics

MCMC diagnostics for the chain can either be based on a small sample of cells, specified by the `MCMCdiag` argument of the `mcmcpars` function, or via the full output from data dumped to disk;

Figure 6: Plot of the relative risk.



the latter is dealt with in Section 3.6.4. The `hvals` command returns the value of h used at each iteration in the algorithm, the left hand plot in Figure 7 shows the values of h for the non-burnin period of the chain, note that the adaptive algorithm was started with $h = 1$, it very quickly converged to around 0.000015.

```
> plot(hvals(lg)[10000:50000],type="l",xlab="Iteration",ylab="h")
> tr <- mcmctrace(lg)
> plot(tr,idx=1:5)
```

Trace plots (the right hand plot of Figure 7) are also available using the function `mcmctrace.lgcpPredict` as in the above code; note that this function returns the trace for Γ. To plot the autocorrelation function, the standard **R** function can be used eg `acf(tr$trace[,1])` gives the acf of the first saved chain.

3.6.4 NetCDF

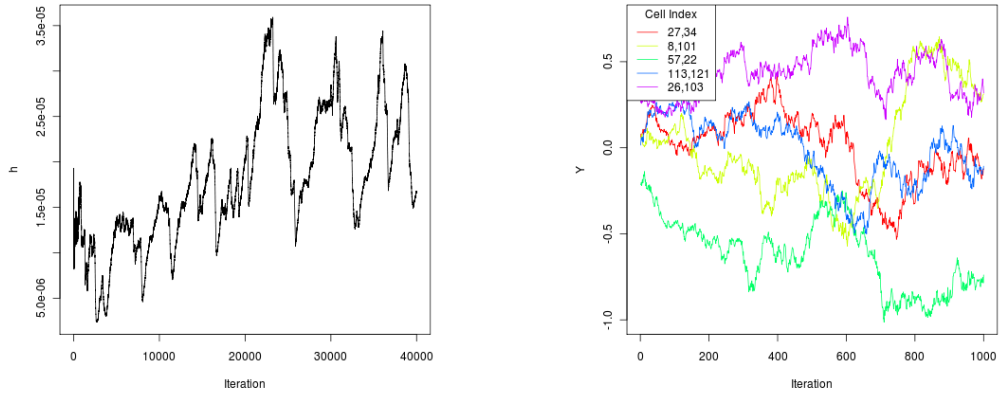
lgcp provides functions for accessing and performing computations on MCMC runs dumped to disk. Due to the possibility of the resulting dataset being very large, **lgcp** uses the cross-platform `NetCDF` file format for storage and rapid data access provided by the package **ncdf** [11]. Access to subsets of data is via a file indexing system, which removes the need to load the complete data into memory.

Subsets of data dumped to disk can be accessed with the `extract` function:

```
> subsamp <- extract(lg,x=c(4,10),y=c(32,35),t=6,s=-1)
```

which returns an array of dimension $7 \times 4 \times 1 \times 1000$. The arguments `x` and `y` refer to the range

Figure 7: MCMC diagnostic plots. Left: plot of values of h taken by the adaptive algorithm. Right: trace plots of the 5 saved chains.



of x and y *indices* of the grid of interest, `t` are the time points of interest (times 45 through 50 were used for prediction, so in this case `t=6` refers to the sixth time point ie. time 50) and `s=-1` stipulates that all simulations are to be returned. Arguments are given as a range or set equal to -1, in which case all of the data in that dimension is returned. This function can also extract MCMC traces from individual cells using `extract(lg,x=37,y=12,t=6)`, for example.

Should the user wish to extract data from a polygonal subregion of the observation window, then this can be achieved with the command

```
> subsamp2 <- extract(lg,inWindow=win2,t=6)
```

where `win2` is a polygonal observation window. Here, `win2` had been selected using the following commands:

```
> plot(window(lg))
> win2 <- loc2poly()
```

the first command plots the observation window and the second command is a wrapper function for the **R** function `locator`. When invoked, `loc2poly()` allows the user to select areas of the observation window manually from the graphics device opened by the first command: the user simply makes a series of left clicks, traversing the required window in a single direction (ie clockwise *or* anticlockwise) finishing the polygon with a right click: the resulting selection is converted into a **spatstat** polygonal `owin` object. Obviously, the user could specify the `extract` argument `inWindow` in a formal manner too.

It is also possible to compute expectations over the target using the stored data (cf. Section D): thus if the user decides that some other summary is of interest than those specified by the option `gridmeans`, then this can easily be computed. The syntax is slightly different, in this case, one would call:

```
> ex <- expectation(obj=lg,fun=exceed)
```

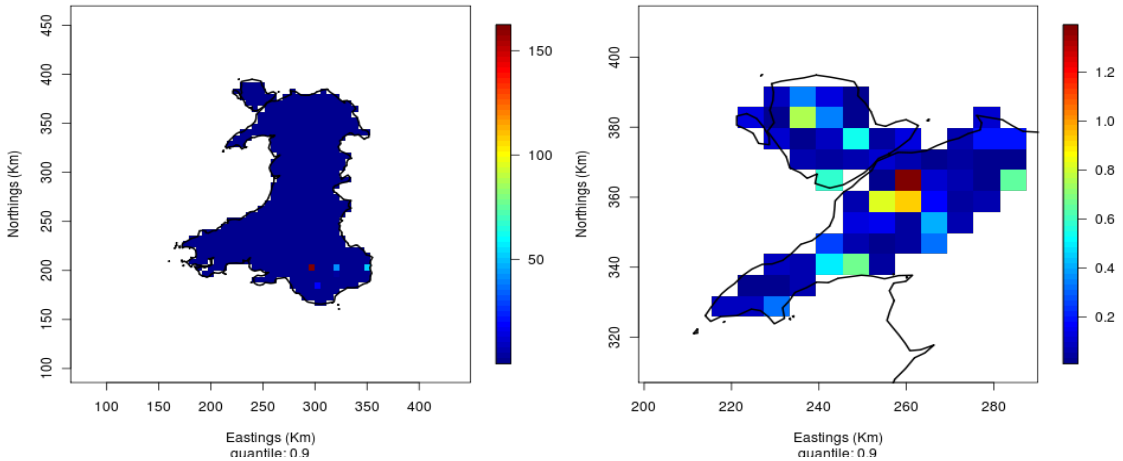
to compute the same exceedances in Section D.

Alternatively, cell-wise quantiles of *functions* of the stored data can also be retrieved and plotted:

```
qt <- quantile(lg,c(0.5,0.75,0.9),fun=exp)
plot(qt,xlab="X coords",ylab="y coords")

qt2 <- quantile(lg,c(0.5,0.75,0.9),fun=exp,inWindow=win2)
plot(qt2,xlab="Longitude",ylab="Latitude")
```

As for the extract function above, quantiles may also be computed for smaller observation windows. The indices of any cells of interest in these plots can be retrieved by typing `identify(lg)`; cells are then selected via left mouse clicks in the graphics device, selection being terminated on a right click.

Figure 8: Plot showing the median (0.5 quantile) of relative risk (obtained using `fun=exp` as in the text) computed from the simulation. Left: Quantiles computed for whole window. Right: Zooming in on the upper left area of the map, representing the areas of Anglesey and the Llŷn Peninsula. Greater detail is available by initially performing the simulation on a finer grid.

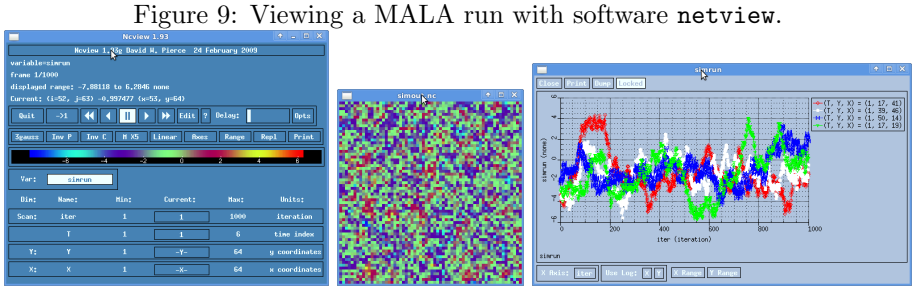


Lastly, Linux users can benefit from the software **Ncview**, available from `http://meteora.ucsd.edu/~pierce/ncview_home_page.html`, which provides fast visualisation of NetCDF files. Figure 9 shows a screenshot, with the control panel (left), an image of one of the sampled grids (middle) and several MCMC chains (right), which are obtained by clicking on the sampled grids; up to 5 chains can be displayed at once. There are equivalent tools for Windows users.

3.6.5 Plotting Exceedance Probabilities

Recall the object `exceed` defined above, this was a function with an attribute giving a vector of thresholds to compute cell-wise probabilities that the relative risk is greater than each of those

Figure 9: Viewing a MALA run with software `netview`.



thresholds. A figure can be produced either directly from the `lgcpPredict` object,

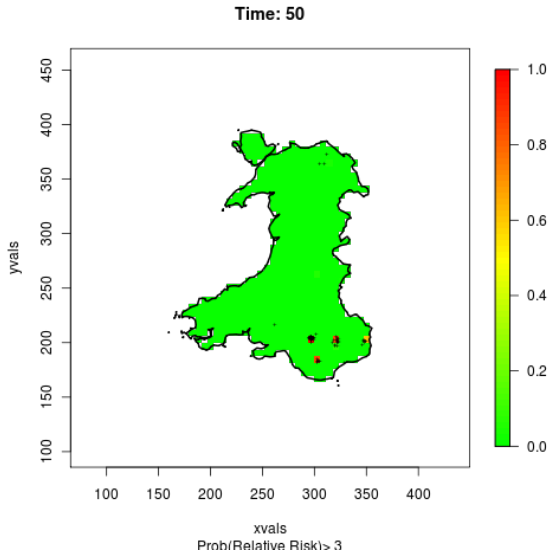
```
> plotExceed(lg, fun = "exceed")
```

or equivalently, from the output of an `expectation` on an object dumped to disk:

```
> plotExceed(ex[[6]], fun = "exceed",lgcppredict=lg)
```

Recall that the option `lastonly=TRUE` was selected for `MonteCarloAverage`, so `ex[[6]]` in the second example above corresponds to the same set of plots as for the first example. The advantage of expectations computed from data dumped to disk is flexibility: if the user now wanted to plot the exceedances for day 6, then this is simply achieved by replacing `ex[[6]]` with `ex[[5]]` and furthermore, exceedances for a completely new set of thresholds can also be computed simply by creating `exceed2 <- exceedProbs(c(2.3,4))` for instance. An example of the output is given in Figure 10.

Figure 10: Plot of exceedance probabilities.



4 Advanced use of lgcp

4.1 Writing Adaptive MCMC schemes

There are two generic functions to consider when writing adaptive MCMC routines: `initialiseAMCMC` and `updateAMCMC`, these respectively define the initialisation and the updating procedures for the adaptive scheme of interest. The task of the user is therefore to tell `lgcpPredict` what value of h to use at iteration 1, and how to update it. **lgcp** has two schemes built in: `constanth` and `andrieuthomsh` detailed below.

4.1.1 A 'Simple' Example: `constanth`

This example shows how the scheme `constanth` was implemented. This is not really an adaptive MCMC scheme and just returns the (fixed) value of h set by the user. In `lgcpPredict`, this 'adaptive' scheme would be specified using `adaptivescheme=constanth(0.01)` in the `mcmc.control` argument and would have the effect of returning $h = 0.01$ at each iteration of the MCMC.

The user is required to write three functions: `constanth`, and for compatibility with the S3 implementation of this framework, `initialiseAMCMC.constanth` and `updateAMCMC.constanth`; these functions are detailed below.

```
constanth <- function(h){
    obj <- h
    class(obj) <- c("constanth","adaptivemcmc")
    return(obj)
}

initialiseAMCMC.constanth <- function(obj,...){
    return(obj)
}

updateAMCMC.constanth <- function(obj,...){
    return(obj)
}
```

When called, the first of these functions creates an object of super-class `constanth`, this is just a numeric with a class attribute attached. The other two functions simply return the value of h specified by the user at appropriate positions in the code `MALAlgcp`.

4.1.2 A More Complex Example: `andrieuthomsh`

The second example shows how to implement a rather neat method of [1]. A Robbins-Munro stochastic approximation update is used to adapt the tuning parameter of the proposal kernel [12]. The idea is to update the tuning parameter at each iteration of the sampler:

$$h^{(i+1)} = h^{(i)} + \eta^{(i+1)}(\alpha^{(i)} - \alpha_{\text{opt}}),$$

where $h(i)$ and $\alpha^{(i)}$ are the tuning parameter and acceptance probability at iteration i and α_{opt} is a target acceptance probability. For Gaussian targets, and in the limit as the dimension of the problem tends to infinity, an appropriate target acceptance probability for MALA algorithms is 0.574 [13]. The sequence $\{\eta^{(i)}\}$ is chosen so that $\sum_{i=0}^{\infty} \eta^{(i)}$ is infinite whilst $\sum_{i=0}^{\infty} \left(\eta^{(i)}\right)^{1+\epsilon}$ is finite for $\epsilon > 0$. These two conditions ensure that any value of h can be reached, but in a way that maintains the ergodic behaviour of the chain. One class of sequences with this property is,

$$\eta^{(i)} = \frac{C}{i^{\alpha}},$$

where $\alpha \in (0, 1]$ and $C > 0$ [1].

An `adaptivescheme` implementing this algorithm must therefore know what value of h to start with, the values of the parameters α and C and the target acceptance probability, hence the choice of arguments for the function `andrieuthomsh` in the code below:

```
andrieuthomsh <- function(inith,alpha,C,targetacceptance=0.574){
    if (alpha<=0 | alpha>1){
        stop("parameter alpha must be in (0,1]")
    }
    if (C<=0){
        stop("parameter C must be positive")
    }
    obj <- list()
    obj$inith <- inith
    obj$alpha <- alpha
    obj$C <- C
    obj$targetacceptance <- targetacceptance

    itno <- 0 # iteration number, gets reset after burnin
    incrit <- function(){
        itno <<- itno + 1
    }
    restit <- function(){
        itno <<- 0
    }
    obj$incritno <- incrit
```

```
    obj$restartit <- restit

    curh <- inith # at iteration 1, the current
                  # value of h is just the initial value
    hupdate <- function(){
        curh <<- exp(log(curh) + (C/(itno^alpha))*
            (get("ac",envir=parent.frame(2))-targetacceptance))
    }
    reth <- function(){
        return(curh)
    }
    obj$updateh <- hupdate
    obj$returncurh <- reth

    class(obj) <- c("andrieuthomsh","adaptivemcmc")
    return(obj)
}
```

This function returns an object of super-class `andrieuthomsh`, which is a list object consisting of the parameters specified by the user and additionally some internal functions that are responsible for the updating. Note that in `updateAMCMC.andrieuthomsh`, the internal functions are simply called, and therefore it is these internal functions that actually define the adaptive scheme. The internal functions, `incrit`, `restit`, `hupdate` and `reth` perform respectively the following tasks: increase the internal iteration counter, restart the internal iteration counter, do the actual updating of h and lastly return the current value of h.

Note that from a developmental point of view, the piece of code,

```
get("ac",envir=parent.frame(2))
```

gets the current acceptance probability, which in `MALAlgcp` is stored as an object called `ac`.

To initialise the scheme, the method for `initialiseAMCMC` simply returns the initial value of h set by the user:

```
initialiseAMCMC.andrieuthomsh <- function(obj,...){
    return(obj$inith)
}
```

In the update step, the internal functions created by the `andrieuthomsh` function are invoked. The procedure is as follows (1) information about the mcmc loop is retrieved using `get("mcmcloop",envir`parent.frame())= then if the algorithm has just come out of the burn in period, the adaptation of h is restarted[1],

[1]This just give h some extra freedom to explore the parameter space as compared to an algorithm that did not restart. Doing this does not affect the convergence of the algorithm

next the internal iteration counter is incremented, and lastly the value of h is updated and returned (the procedures for these internal functions are printed above in the code for `andrieuthomsh`).

```
updateAMCMC.andrieuthomsh <- function(obj,...){
    mLoop <- get("mcmcloop",envir=parent.frame())
    if(iteration(mLoop)==(mLoop$burnin)+1){
        obj$restartit() # adaptation of h restarted after burnin
    }
    obj$incritno() # this line must appear below the above four lines
    obj$updateh()
    return(obj$returncurh())
}
```

A Spatio-temporal Log Gaussian Cox Processes

Let $W \subset \mathbb{R}^2$ be an observation window in space and $T \subset \mathbb{R}_{\geq 0}$ be an interval of time of interest. Cases occur at spatio-temporal positions $(x,t) \in W \times T$ according to an inhomogeneous spatio-temporal Cox process, i.e. a Poisson process with a stochastic intensity $R(x,t)$, The number of cases, $X_{S,[t_1,t_2]}$, arising in any $S \subseteq W$ during the interval $[t_1,t_2] \subseteq T$ is then Poisson distributed conditional on $R(\cdot)$,

$$X_{S,[t_1,t_2]} \sim \text{Poisson}\left\{\int_S \int_{t_1}^{t_2} R(s,t)\mathrm{d}s\mathrm{d}t\right\}. \tag{1}$$

Following [7], the intensity is decomposed multiplicatively as

$$R(s,t) = \lambda(s)\mu(t)\exp\{\mathcal{Y}(s,t)\}. \tag{2}$$

In (2),the *fixed spatial component*, $\lambda : \mathbb{R}^2 \mapsto \mathbb{R}_{\geq 0}$, is a known function, proportional to the population at risk at each point in space and scaled so that

$$\int_W \lambda(s)\mathrm{d}s = 1, \tag{3}$$

whilst the *fixed temporal component*, $\mu : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$, is also a known function with,

$$\mu(t) = \lim_{\delta t \to 0}\left\{\frac{\mathbb{E}[X_{W,\delta t}]}{|\delta t|}\right\}. \tag{4}$$

The function $\mathcal{Y}$ is a Gaussian process, continuous in both space and time. In the nomenclature of epidemiology, the components λ and μ determine the *endemic* spatial and temporal component of the population at risk; whereas $\mathcal{Y}$ captures the residual variation, or the *epidemic* component.

The Gaussian process, $\mathcal{Y}$, is second order stationary with minimally-parametrised covariance function,

$$\text{cov}[\mathcal{Y}(s_1,t_1),\mathcal{Y}(s_2,t_2)] = \sigma^2 r(||s_2 - s_1||;\phi)\exp\{-\beta(t_2 - t_1)\},$$

where $||\cdot||$ is a suitable norm on $\mathbb{R}^2$, for instance the Euclidean norm, and $\sigma, \phi, \beta > 0$ are known parameters. In the **lgcp** package, the isotropic spatial correlation function, r, may take one of several forms and possibly require additional parameters. The parameter σ scales the log-intensity, whilst the parameters ϕ and β govern the rates at which the correlation function decreases in space and in time, respectively. The mean of the process $\mathcal{Y}$ is set equal to $-\sigma^2/2$ so as to give $\mathbb{E}[\exp\{Y\}] = 1$, hence the endemic/epidemic analogy above.

B Inference

As in [10], [6] and [7], this article considers a discretised version of the above model defined on a regular grid over space and time. Observations, X are therefore treated as cell counts on this grid, instead of point-instances. The discrete version of $\mathcal{Y}$ on this grid will be denoted Y; since Y is a finite collection of random variables, the properties of $\mathcal{Y}$ imply that Y has a multivariate Gaussian density with covariance matrix $\Sigma = \sigma^2 r(||\cdot||;\phi)\exp\{-\beta\Delta t\}$. The correlation function r is evaluated on the centroids of the grid cells and Δt denotes the time over which the observation occurred.

Let X_t denote an observation over the spatial grid at time t and $X_{t_1:t_2}$ denote the observations at times $t_1, t_1+1, \ldots, t_2$. Ideally, inference would concern the 'current' latent field, Y_t, given the observations to date, $X_{1:t}$, but unfortunately, due to a high-dimensional integral, the density of $\pi(Y_t|X_{1:t})$ is not available analytically. It *is* however possible to infer the joint density of a collection, $Y_{1:t}$ given $X_{1:t}$ for instance. Since simulation from $\pi(Y_{1:t}|X_{1:t})$ is generally computationally very intensive, if interest is in the state of the system at time t_2, an alternative is to sample from $Y_{t_1:t_2}$ given $X_{t_1:t_2}$ for some suitable $t_1 > 1$ and fairly close to t_2. This approach is justified in [6], the argument being that observations from a certain distance in the past do not affect inference for the present. The aim in this article is therefore infer the state of the underlying field given the observations of interest,

$$\pi(Y_{t_1:t_2}|X_{t_1:t_2}) \propto \pi(X_{t_1:t_2}|Y_{t_1:t_2})\pi(Y_{t_1:t_2}). \tag{5}$$

In order to evaluate $\pi(Y_{t_1:t_2})$ in the above, the parameters of the process Y must either be known or estimated from the data. Estimation of σ, ϕ and θ may be achieved either in a full Bayesian framework, or by approximate methods. The methods implemeted in current version of the **lgcp** package fall into the latter category and a detailed description of these is given in [6] and [7]. Briefly, approximate estimation of the spatial parameters σ and ϕ involves matching empirical and theoretical estimates of the inhomogeneous K-function, or g function, as in [2]. For the parameter θ, the idea is to match the theoretical and empirical autocovariance function of the total case-counts per time interval. Having estimated the parameters of the process Y, the investigator can now proceed to plug-in-prediction.

B.1 Discretising and the Fast-Fourier Transform

The first barrier to inference is computation of the covariance matrix, Σ, which even for relatively coarse grids is very large. Fortunately, for regular spatial grids of size $2^m \times 2^n$, there exist fast methods for computing this based on the discrete fast Fourier transform (FFT), see [18]. The general idea is to embed Σ in a symmetric circulant matrix, $C = Q\Lambda Q^*$, where Λ is a diagonal matrix of eigenvalues of C and Q is a unitary matrix (* denotes the Hermitian transpose). The entries of Q are given by the discrete Fourier transform. Computation of $C^{1/2}$, which is useful for both simulation and evaluation of the density of Y, is then straightforward for the properties of Q imply,

$$C^{1/2} = Q\Lambda^{1/2}Q^*.$$

In fact, Monte Carlo simulation from $\pi(Y_{t_1:t_2}|X_{t_1:t_2})$, which is to be discussed in the next section is greatly improved by working with a linear transformation of Y, partially determined by the matrix C. **lgcp** returns results on a grid of size $M \times N \equiv 2^m \times 2^n$ for some positive integers m and n; this is extended to a grid of size $2M \times 2N$ for computation.

B.2 The Metropolis-Adjusted Langevin Algorithm

Writing $\Gamma_t = \Lambda^{-1/2}Q(Y_t - \mu)$, the target of interest is given by,

$$\pi(\Gamma_{t_1:t_2}|X_{t_1:t_2}) \propto \left[\prod_{t=t_1}^{t_2} \pi(X_t|Y_t)\right]\left[\pi(\Gamma_{t_1})\prod_{t=t_1+1}^{t_2} \pi(\Gamma_t|\Gamma_{t-1})\right], \tag{6}$$

where the first term on the right hand side of (6) corresponds to the first bracketed term on the right hand side of (5) and the second bracketed term is the joint density, $\log\{\pi(\Gamma_{t_1:t_2})\}$. Being an Ornstein-Uhlenbeck process in time, the transition density of $\pi(\Gamma_t|\Gamma_{t-1})$ admits an exact solution and is just a normal density, see [6].

Since the gradient of the above can also be written down, a natural and efficient method for simulating from this density is via Metropolis-Hastings with a Langevin-type proposal,

$$q(\Gamma, \Gamma') = \mathcal{N}\left[\Gamma'; \Gamma + \frac{1}{2}\nabla \log\{\pi(\Gamma|X)\}, h^2\mathbb{I}\right],$$

where $\mathcal{N}(y; m, v)$ is a Gaussian density evaluated at y with mean m and variance v, $\mathbb{I}$ is the identity matrix and $h > 0$ is a scaling parameter [9, 8].

Various theoretical results exist concerning the optimal acceptance probability of the MALA (Metropolis-Adjusted Langevin Algorithm) – see [14] and [13] for example. In practical applications, the target acceptance probability is often set to 0.574, which would be approximately optimal for a Gaussian target as the dimension of the problem tends to infinity. An algorithm for the automatic choice of h, so that this acceptance probability is achieved without disturbing the ergodic property of the chain, is also straightforward to implement, see [1].

C Adaptive MCMC

The MALA proposal tuning parameter h in Appendix B.2 must also be chosen. The most straightforward way to do this is to set `adaptivescheme=constanth(0.001)`, this gives $h = 0.001$ for example. Without a *lengthy* tuning process, the 'best h' (ie. the h that leads to the best mixing of the algorithm) is not known. One solution to the problem of having to choose a scaling parameter via pilot runs is to use adaptive MCMC (AMCMC - see [15] and [1]). The main idea of adaptive algorithms is to use information from the MCMC chain to help choose tuning parameters for the proposal kernel. However, since the proposal kernel then depends on the history of the chain, the Markov property is no longer satisfied and some care must be taken to ensure that the correct ergodic distribution is preserved.

An elegant method, introduced by [1] uses a Robbins-Munro stochastic approximation update to adapt the tuning parameter of the proposal kernel [12]. The idea is to update the tuning parameter at each iteration of the sampler:

$$h^{(i+1)} = h^{(i)} + \eta^{(i+1)}(\alpha^{(i)} - \alpha_{\text{opt}}),$$

where $h(i)$ and $\alpha^{(i)}$ are the tuning parameter and acceptance probability at iteration i and α_{opt} is a target acceptance probability. For Gaussian targets, and in the limit as the dimension of the problem tends to infinity, an appropriate target acceptance probability for MALA algorithms is 0.574 [13]. The sequence $\{\eta^{(i)}\}$ is chosen so that $\sum_{i=0}^{\infty} \eta^{(i)}$ is infinite whilst $\sum_{i=0}^{\infty} \left(\eta^{(i)}\right)^{1+\epsilon}$ is finite for $\epsilon > 0$. These two conditions ensure that any value of h can be reached, but in a way that maintains the ergodic behaviour of the chain. One class of sequences with this property is,

$$\eta^{(i)} = \frac{C}{i^{\alpha}},$$

where $\alpha \in (0, 1]$ and $C > 0$ [1].

The tuning constants for this algorithm are set with the function `andrieuthomsh`.

```
> args(andrieuthomsh)
function (inith, alpha, C, targetacceptance = 0.574)
```

In the above, `inith` is the initial value of h to start out from; the remaining arguments correspond to their counterparts in the text above. It is also possible for the advanced user to write their own adaptive scheme (see Section 4.1).

D Monte Carlo Averaging and Exceedance Probabilities

It is also of interest to be able to compute Monte Carlo expectations,

$$\begin{aligned}\mathbb{E}_{\pi(Y_{t_1:t_2}|X_{t_1:t_2})}[g(Y_{t_1:t_2})] &= \int_W g(Y_{t_1:t_2})\pi(Y_{t_1:t_2}|X_{t_1:t_2})\mathrm{d}Y_{t_1:t_2}, \\ &\approx \frac{1}{n}\sum_{i=1}^{n} g(Y_{t_1:t_2}^{(i)})\end{aligned}$$

where g is a function of interest, $Y_{t_1:t_2}^{(i)}$ is the ith retained sample from the target and n is the total number of retained iterations. For example, to compute the mean of $Y_{t_1:t_2}$ set,

$$g(Y_{t_1:t_2}) = Y_{t_1:t_2},$$

the output from such a Monte Carlo average would be a set of $t_2 - t_1$ grids, each cell of which being equal to the mean over all retained iterations of the algorithm.

An example in epidemiological studies are exceedance probabilities, see [7]. These give the cell-wise probability of the relative risk exceeding certain thresholds, for example, it may be of clinical interest to know if any of the relative risk has exceeded 2 at any location in W. In mathematics, this is $\mathbb{P}[\exp(Y) > k]$ for some threshold k; it turns out that such quantities can also be expressed as Monte Carlo expectations, hence the user may also compute these online using `MonteCarloAverages`. The probability is written,

$$\mathbb{P}[\exp(Y_{t_1:t_2}) > k] = \mathbb{E}_{\pi(Y_{t_1:t_2}|X_{t_1:t_2})}[\mathbb{I}(Y_{t_1:t_2} > k)] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(Y_{t_1:t_2}^{(i)} > k),$$

where $\mathbb{I}$ is the indicator function. In the notation above, this amounts to,

$$g(Y_{t_1:t_2}) = \mathbb{I}(Y_{t_1:t_2} > k).$$

Exceedance probabilities are provided as part of **lgcp** in the function `exceedProbs`.

```
> exceedProbs

function(threshold){
    fun <- function(Y){
        EY <- exp(Y)
        d <- dim(Y)
        len <- length(threshold)
        if(len==1){
            return(matrix(as.numeric(EY>threshold),d[1],d[2]))
        }
        else{
            A <- array(dim=c(d[1],d[2],len))
```

```
            for(i in 1:len){
                A[,,i] <- matrix(as.numeric(EY>threshold[i]),d[1],d[2])
            }
            return(A)
        }
    }
    attr(fun,"threshold") <- threshold
    return(fun)
}
```

The user must first decide on the thresholds of interest (eg 1.5, 2 and 3 say) and then create a function to compute the required exceedances:

```
exceedfunc <- exceedProbs(c(1.5,2,3))
```

the object `exceed` is now a function that returns the exceedance probabilities as an array object of dimension $M \times N \times 3$, where the output grid is of size $M \times N$ (ie. on an FFT grid of size $2M \times 2N$). This function can be passed through to the `gridmeans` option via `MonteCarloAverage(c("gfun","exceed")`, so that alongside the mean, the exceedance probabilities are also returned.

E Rotation

As mentioned above, the MALA algorithm works on a regular square grid placed over the observation window. In model fitting, the user will be responsible for providing a physical grid size on which to perform estimation/prediction; the gridded observation window is then extended or shrunk automatically to obtain a $2^M \times 2^N$ grid on which the simulation happens. By default, the orientation of this 'extended' grid is the same as the object `win`. If the observation window is elongated and set at a diagonal, then some loss of efficiency occurs as the size of the extended grid is larger than it would be if the polygonal window was first rotated in line with the axes, the estimation then taking place in the rotated space.

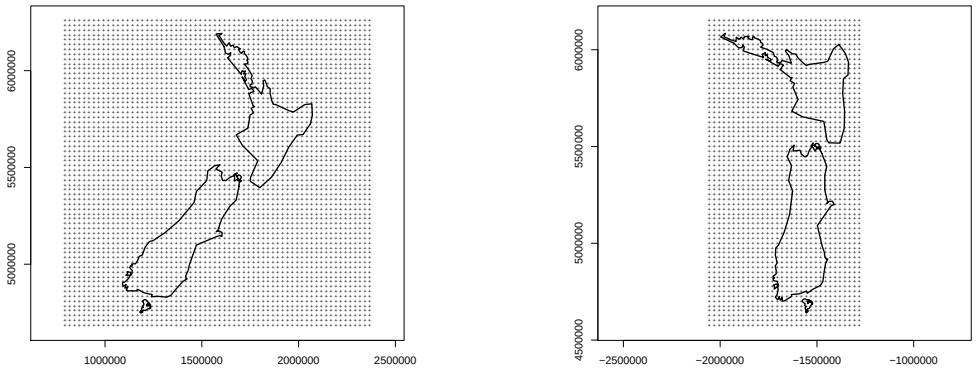
To illustrate this point, suppose `xyt3` is an `stppp` object with such an elongated and diagonal window. The function `roteffgain` displays whether any efficiency can be gained by rotation; clearly this not only depends on the observation window, but also on the size of the square cells on which the analysis will be performed. In the example below, the user wishes to perform the analysis using a cell width of 25km (corresponding to `gridsize=25000` in the code below):

```
> roteffgain(xyt3,gridsize=25000)
By rotating observation window, the efficiency gain would be: 200%,
   see ?getRotation.stppp
NOTE: efficiency gain is measured as the percentage increase in FFT
   grid cells from not rotating compared with rotating
[1] TRUE
```

The routine simply return `FALSE` if there is no 'efficiency gain'. Note that the efficiency gain is *not* a reflection on computational speed, but rather a measure of how many fewer cells the MALA is required to estimate; this is illustrated in Figure 11. As a technical aside, a better measure would be a ratio of mixing times for the MCMC chains based on un-rotated and rotated windows; however, as the mixing time depends on how well the MALA has been tuned, it is not clear how this can be accurately estimated in a straightforward manner.

Having ascertained whether rotation is advantageous, the optimally rotated data, observation window and rotation matrix may be obtained using the function `getRotation`. For the purposes of fitting the model using `lgcpPredict`, there is also an `autorotate` option: this allows the user to perform MALA on a rotated grid with minimal input so long as rotation leads to a gain in efficiency. If the model is fitted using a rotated frame, then the predictions will also be returned in this frame: this means that in the original orientation the output will be on a grid of diamond-shaped cells, rather than on a grid of square cells. **lgcp** provides methods for the generic function `affine` so that `stppp` and `spatialAtRisk` objects can be rotated manually.

Figure 11: Illustrating the potential gain in efficiency by rotating the observation window. Left plot: the selected grid without rotation. Right plot: the optimal rotated grid.



F Gradient Truncation

One unfortunate property of the Metropolis adjusted Langevin algorithm is that the chain is prone to taking very long excursions from the mode of the target; this behaviour can have a detrimental effect on the mixing of the chain and consequently on any results. The tendency to make long excursions is caused by instability in the computation of the gradient vector, but the issue is relatively straightforward to rectify without affecting convergence properties [10]. The key is to truncate the gradient vector if it becomes too large. If `gradtrunc=NULL`, then an appropriate truncation is automatically selected by the code.

As far as the authors are aware, there are no guidelines for selecting this truncation parameter in the literature, but the method employed in this version of the package takes the maximum achieved gradient over a set of 100 independent realisations of $\Gamma_{t_1:t_2}$.

References

[1] Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18(4):343–373, 2008.

[2] A.˜J. Baddeley, J.˜Møller, and R.˜Waagepetersen. Non-and semi-parametric estimation of interaction in inhomogeneous point patterns. *Statistica Neerlandica*, 54:329–350, 2000.

[3] Adrian Baddeley and Rolf Turner. spatstat: An r package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005.

[4] Roger˜S. Bivand, Edzer˜J. Pebesma, and Virgilio Gomez-Rubio. *Applied spatial data analysis with R*. Springer, NY, 2008.

[5] Adrian Bowman, Ewan Crawford, Gavin Alexander, and Richard˜W. Bowman. rpanel: Simple interactive controls for R functions using the tcltk package. *Journal of Statistical Software*, 17(9):1–18, 2007.

[6] A.˜Brix and P.˜J. Diggle. Spatiotemporal prediction for log-gaussian cox processes. *Journal of the Royal Statistical Society, Series B*, 63(4):823–841, 2001.

[7] Peter Diggle, Barry Rowlingson, and Ting-li Su. Point process methodology for on-line spatio-temporal disease surveillance. *Environmetrics*, 16(5):423–434, 2005.

[8] W.˜K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[9] N.˜Metropolis, A.˜W. Rosenbluth, M.˜N. Rosenbluth, A.˜H. Teller, and E.˜Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[10] Jesper Møller, Anne˜Randi Syversveen, and Rasmus˜Plenge Waagepetersen. Log Gaussian Cox processes. *Scandinavian Journal of Statistics*, 25(3):451–482, 1998.

[11] David Pierce. ncdf home page: A netcdf package for r, 2011.

[12] H˜Robbins and S.˜Munro. A stochastic approximation methods. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[13] G.˜Roberts and J.˜Rosenthal. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science*, 16(4):351–367, 2001.

[14] Gareth˜O. Roberts and Jeffrey˜S. Rosenthal. Optimal scaling of discrete approximations to langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60:255–268(14), 1998.

[15] Gareth˜O. Roberts and Jeffrey˜S. Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, June 2009.

[16] Martin Schlather. Simulation and analysis of random fields, 2001.

[17] B.˜Taylor, T.˜Davies, B.˜Rowlingson, and P.˜Diggle. lgcp – an R package for inference with spatiotemporal log-Gaussian Cox processes, 2011. In Preparation.

[18] Andrew T.˜A. Wood and Grace Chan. Simulation of stationary gaussian processes in [0,1]d. *Journal of Computational and Graphical Statistics*, 3(4):409–432, 1994.