Supplementary Material

sppmix: Poisson point process modeling using normal mixture models

Athanasios C. Micheas and Jiaxun Chen

Department of Statistics

University of Missouri

April 19, 2018

1 The sppmix package

The sppmix package can be found at the CRAN site \url{https://CRAN.R-project.org/package=sppmix}. A list of all sppmix package tutorials can be found at \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_tutorial_links.html}. Use the sppmix package function Demosppmix to run a package demo or open the vignettes in your browser. Currently we have included the following tutorials (vignettes):

1. Introduction to sppmix objects: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_all_objects_intro.html}

2. Basic Objects Examples: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_basic_objects_examples.html}

3. Plotting in sppmix: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_plotting.html}

4. IPPP Model fitting using DAMCMC, no edge effects: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_DAMCMCunbounded.html}

5. IPPP Model fitting using DAMCMC, edge effects: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_DAMCMCbounded.html}

6. IPPP Model fitting using BDMCMC, no edge effects: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_BDMCMCunbounded.html}

7. IPPP Model fitting using BDMCMC, edge effects: \url{http://faculty.missouri.edu/~micheasa/sppmix/sppmix_BDMCMCbounded.html}


10. Model diagnostics: Label Switching: [http://faculty.missouri.edu/~micheasa/sppmix/sppmix_diagnose_label_switching.html](http://faculty.missouri.edu/~micheasa/sppmix/sppmix_diagnose_label_switching.html)

11. Marked IPPP Model fitting via conditioning: [http://faculty.missouri.edu/~micheasa/sppmix/sppmix_MIPPP_cond.html](http://faculty.missouri.edu/~micheasa/sppmix/sppmix_MIPPP_cond.html)

We begin by describing the main package objects. Specific examples on all these objects and specific tasks, such as, checking and fixing label switching, performing model diagnostics, model selection and checking are given in the examples that follow in the next section.

### 1.1 Main sppmix package objects

There are several objects introduced in the sppmix package. These objects are utilized in many tasks, from building basic models, to model fitting and plotting. We study the specifics of these objects in later sections. Here we collect the main objects and the lists of functions that operate on them. The list of objects includes the

- **normmix object**: this is an object that represents a 2d mixture with normal components. Note that several functions can be applied to this object or return this type of object and can be used for additional visualization and operations on the created normal mixture. Important members of this object include the parameters defining the mixture: $m$, $ps$, $mus$, and $sigmas$. Functions operating on the normmix object include:
  - plot: generic 3d plotting of the density,
  - summary: basic information about the normmix object,
  - rnormmix: can be used to create normal mixtures by sampling the mixture parameters randomly,
  - approx_normmix: approximates the mass of each component within the given window,
  - dnormmix: computes the density of the mixture within the given window, which can then be used for 2d and 3d plotting,
  - plotmix_2d: can be used to plot the density in 2d (elevation or contour plot), and
  - to_int_surf: creates an intensity_surface object from a given mixture.

- **intensity_surface object**: this is an object that represents the intensity surface of an Inhomogeneous Poisson point process (IPPPP). Important members of this object include the parameters defining the mixture: $m$, $ps$, $mus$, $sigmas$, and in addition $lambda$ and $window$. Functions that work with the intensity_surface object include:
  - plot: generic 3d plotting of the intensity surface,
  - summary: basic information about the intensity_surface object,
- `rmixsurf`: creates an IPPP mixture intensity surface on the given 2d window, with the component means, covariances and probabilities chosen randomly,
- `plotmix_2d`: can be used to plot the intensity surface in 2d (elevation or contour plot),
- `dnormmix`: computes the intensity values for the mixture intensity within the given window, which can then be used for 2d and 3d plotting,
- `to_int_surf`: allows us to change the parameter $\lambda$ and the window of the intensity_surface object,
- `CompareSurfs`: computes distance measures for quantifying the difference between two surfaces,
- `ktest2dsurf`: performs a two-dimensional Kolmogorov-Smirnov goodness-of-fit test for a point pattern against a given intensity surface,
- `mc_gof`: performs a Monte Carlo test of goodness-of-fit for a given point pattern against a given mixture intensity surface,
- `GetIPPPLikValue`: given a point pattern and a surface this function calculates the IPPP likelihood value,
- `rMIPPP_cond_loc`: given a point pattern and a surface this function generates a Marked Poisson point process (conditional on location).
- `rMIPPP_cond_mark`: given a point pattern and a surface this function generates a Marked Poisson point process (conditional on discrete mark).

- `sppmix object`: this object represents a generated point pattern from an IPPP with mixture intensity surface. This object is the result of a call to the function `rsppmix`, which generates point patterns over a window, based on an intensity surface described by an intensity_surface object. Important members of this object include $x$: a vector of x coordinates of the events, $y$: a vector of y coordinates of the events, $n$: the number of events, $window$: the window of observation (an object of class spatstat::owin), $marks$: optional vector or data frame of marks, and $comp$: vector of true allocation variables. Under edge effects, the points can be truncated to be within the window. The `sppmix` object is equivalent in almost all its properties to the `ppp` object from the spatstat R package. In addition, it has a member variable named `comp`, which contains the true membership indicators (allocation variables). These variables indicate from which mixture component a point in the point pattern comes from. Functions that work with the `sppmix` object include:
  - `plot2dPP`: generic 2d plotting of the point pattern,
  - `summary`: basic information about the `sppmix` object,
  - `rsppmix`: generates a point pattern from an IPPP with a given mixture intensity surface,
  - `Count_pts`: counts the number of points from the point pattern within a specified window,
  - `plot_true_labels`: plots the true membership indicators (or allocation variables) of each point of the point pattern to one of the mixture components,
  - `plotmix_2d`: can be used to plot the point pattern in 2d along with an intensity surface,
  - `est_mix_damcmc`: using the point pattern, we fit an IPPP with mixture intensity using DAM-CMC for a fixed number of components,
- `est_mix_bdmcmc`: using the point pattern, we fit an IPPP with mixture intensity using BD-MCMC for a random number of components,
- `est_intensity_np`: using the point pattern and under an Epanechnikov kernel we calculate an estimate of intensity surface while accounting for edge effects,
- `ktest2d`: performs a two-dimensional Kolmogorov-Smirnov goodness-of-fit test on two point patterns,
- `ktest2dsurf`: performs a two-dimensional Kolmogorov-Smirnov goodness-of-fit test for a point pattern against a given intensity surface,
- `mc_gof`: performs a Monte Carlo test of goodness-of-fit for a given point pattern against a given mixture intensity surface,
- `GetIPPLikValue`: given a point pattern and a surface this function calculates the IPPP likelihood value,
- `Get_Rdiag`: retrieves the Gelman-Rubin convergence diagnostic,
- `plot_MPP_fields`: plot the mark probability fields (if the object has mark values in the `marks` component),
- `rMIPPP_cond_loc`: given a point pattern and a surface this function generates a Marked Poisson point process (conditional on location),
- `rMIPPP_cond_mark`: given a point pattern and a surface this function generates a Marked Poisson point process (conditional on discrete mark),
- `selectMix`: suggests the best number of components by computing model selection criteria, including AIC (Akaike Information Criterion), BIC (Bayesian Information Criterion), ICLC (Integrated Classification Likelihood Criterion).

- `damcmc_res` object: this object represents the results of a call to the function `est_mix_damcmc`, which fits an IPPP with mixture intensity via the Data Augmentation MCMC (DAMCMC) algorithm. Please see the help page on function `est_mix_damcmc` for a full list of members in this object. The following functions can be used with this object:
- `est_mix_damcmc`: using a point pattern we fit an IPPP with mixture intensity using DAMCMC for a fixed number of components and return a `damcmc_res` object,
- `print`: basic info about the `damcmc_res` object,
- `summary`: retrieves the posterior means of the mixture model parameters along with their credible sets and displays them,
- `plot`: a simple plot call of the `damcmc_res` object will produce a plethora of graphical and numerical descriptions regarding the IPPP fit (these plots can be produced individually by calling the appropriate functions, see below),
- `plot_chains`: plots the MCMC chains for all component means and probabilities,
- `plot_ind`: plots the posterior means of the membership indicators (or allocation variables) of each point to one of the mixture components, based on the DAMCMC `damcmc_res` object,
- `plot_avgsurf`: calculates the intensity surface at each posterior realization and then computes the average for the intensity surface over a fine grid. The result is a much smoother posterior estimator of the intensity surface, which is not necessarily the same as the surface of posterior means, which is obtained by `GetPMEst`,
- GetPMEst: this function first calculates the posterior means of the parameters of the components of the mixture intensity, based on the damcmc_res object. Then the surface of posterior means is calculated using the posterior means of the parameters,

- GetMAPEst: calculates the MAP estimates of the parameters of the components of the mixture intensity and returns the corresponding intensity surface,

- GetKLEst: retrieves the surface of Kullback-Leibler (KL) estimates of the parameters of the components of the mixture intensity and returns the corresponding intensity surface,

- check_labels: checks if there is label switching present in the posterior realizations using the chains for the \( \mu \)'s of the damcmc_res object,

- FixLS_da: permutes the posterior realizations of the damcmc_res object in order to fix the labels by either applying an identifiability constraint or by minimizing the squared error loss to find the best permutation,

- drop_realization: drops realizations from the DAMCMC fit and returns the resulting damcmc_res object,

- GetMAPLabels: returns the MAP estimates of the component labels (membership indicator variables) based on a damcmc_res object,

- GetDensityValues: operates on the point pattern and the realizations of a DAMCMC fit (object damcmc_res) and returns a plethora of information about the fit, include values for the marginal, log-likelihood, component densities and more,

- plot_convdiags: based on a ‘damcmc_res’ object, this function will produce many graphs to help assess convergence visually, including running mean plots and autocorrelation plots for all the parameters.

- bdmcmc_res object: this is an object that represents the results of a call to the function est_mix_bdmcmc, which fits an IPPP with mixture intensity via the Birth-Death MCMC (BDMCMC) algorithm. Please see the help page on function est_mix_bdmcmc for a full list of members in this object. All the functions operating on the damcmc_res can be used with a bdmcmc_res and in many cases additional information is displayed; more precisely, using the bdmcmc_res object we obtain the realizations corresponding to the MAP estimator for the number of components, which results in an object of class damcmc_res In addition, the following functions can be used to create and manipulate this object:

  - est_mix_bdmcmc: using a point pattern we fit an IPPP with mixture intensity using BDMCMC for a random number of components and return a bdmcmc_res object,
  - print: basic info about the bdmcmc_res object, including the number of iterations and the frequency table for the distribution of the number of components,
  - summary: retrieves the posterior means of the mixture model parameters along with their credible sets for a specific number of components and displays them,
  - plot: a simple plot call of the bdmcmc_res object will produce a plethora of graphical and numerical descriptions regarding the IPPP fit, including the distribution and chain for the number of components, and the chains and intensity surface plots for the MAP number of components (these plots can be produced individually by calling the appropriate functions, see below),
- plot_CompDist: produces two plots; the trace plot for the number of components based on all realizations of the BDMCMC fit, and a barplot describing the distribution of the number of components,

- plot_chains: plots the MCMC chains for all component means and probabilities for the mixture intensity with MAP number of components (if you want the chains for a specific number of components, extract those realizations first and then use plot_chains on the resulting damcmc_res object),

- plot_ind: plots the posterior means of the membership indicators (or allocation variables) of each point to one of the mixture components, for the mixture intensity with MAP number of components (if you want to plot the posterior means of the indicators for a specific number of components, extract those realizations first and then use plot_ind on the resulting damcmc_res),

- plot_avgsurf: calculates the intensity surface at each posterior realization and then computes the average for the intensity surface over a fine grid, for the mixture intensity with MAP number of components (if you want to plot the posterior means of the indicators for a specific number of components, extract those realizations first and then use plot_avgsurf on the resulting damcmc_res),

- GetPMEst: this function retrieves the intensity surface of posterior means for a requested number of components,

- GetMAPEst: retrieves the MAP estimates of the parameters of the components of the mixture intensity for a requested number of components,

- GetKLEst: retrieves the surface of Kullback-Leibler (KL) estimates of the parameters of the components of the mixture intensity and returns the corresponding intensity surface,

- check_labels: checks if there is label switching present in the posterior realizations using the chains for the $\mu$'s for the mixture intensity with MAP number of components (if you want to plot the posterior means of the indicators for a specific number of components, extract those realizations first and then use check_labels on the resulting damcmc_res),

- FixLS_da: permutes the posterior realizations for the mixture intensity with MAP number of components in order to fix the labels by either applying an identifiability constraint or by minimizing the squared error loss to find the best permutation,

- GetBDTable: retrieves the MAP estimate (mode of the posterior) and the frequency table for the number of components, based on a BDMCMC fit,

- GetBDCompfit: obtain the realizations (as a damcmc_res object) and the corresponding surface of posterior means, for a specific number of components based on a BDMCMC fit,

- drop_realization: drops realizations from the BDMCMC fit and returns the resulting bdmcmc_res object,

- GetBMA: uses the posterior realizations to compute the Bayesian Model Average (BMA) across different number of components and returns the fitted Poisson point process with mixture of normals intensity surface,

- GetMAPLabels: returns the MAP estimates of the component labels (membership indicator variables) based on a bdmcmc_res object and the chain corresponding to MAP number of components,
- GetDensityValues: operates on the point pattern and the realizations of a BDMCMC fit (object bdmcmc_res) and returns a plethora of information about the fit, include values for the marginal, log-likelihood, component densities and more,
- plot_convdiags: based on a ‘damcmc_res’ object, this function will produce many graphs to help assess convergence visually, including running mean plots and autocorrelation plots for all the parameters.

- MIPPP_fit object: this object represents the results of a call to the function est_MIPPP_cond_loc (please see the help page of this function for a full list of members of the MIPPP_fit object), which fits a Marked IPPP (MIPPP) using conditioning, with the ground (locations process) being an IPPP with mixture intensity independent of the marks and the mark distribution depends on the locations of the events. We can use the following functions with this object:
  - rMIPPP_cond_loc: Generate a Marked Poisson point process (conditional on location),
  - rMIPPP_cond_mark: Generate a Marked Poisson point process (conditional on mark),
  - est_MIPPP_cond_loc: fits a MIPPP conditional on locations and returns a MIPPP_fit object,
  - est_MIPPP_cond_mark: fits a MIPPP conditional on marks and returns a MIPPP_fit object,
  - summary: displays important information about the created object, such as, summaries about the marks and the model parameters,
  - plot: produces plots of the marks and their probability fields, as well as, the intensity surface plot for the ground IPPP,
  - plot_MPP_probs: plot the mark probabilities of a marked point pattern (discrete marks only).

1.2 Tasks and corresponding functions

We summarize here the functions in the sppmix package depending on what we need to accomplish. In this way the user will be able to quickly access information on what they are interested in and then they can use the function help page for additional details on how to complete a specific task.

Table 1 lists functions that simulate important objects used to create and model IPPPs and MIPPPs. Table 2 presents functions that can be used to create and manipulate mixtures of normal components that can be later used to model the intensity surface of an IPPP. In table 3 we list functions used to fit an IPPP or MIPPP model, whereas, table 4 presents functions that can be used to process a model fit output for an IPPP or MIPPP model. Functions that help perform model diagnostics are presented in table 5 including how to assess convergence of a model fit, perform goodness of fit tests, model selection and model comparison for an IPPP or MIPPP model. Finally, table 6 lists functions used to visualize created objects and model fits, visualize the USA states or counties and manipulate plots in general.
### Simulation Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Task Accomplished</th>
</tr>
</thead>
<tbody>
<tr>
<td>rnormmix</td>
<td>Generate a mixture with normal components</td>
</tr>
<tr>
<td>rmixsurf</td>
<td>Generate a Poisson process surface object</td>
</tr>
<tr>
<td>rsppmix</td>
<td>Generate a point pattern from a Poisson process</td>
</tr>
<tr>
<td>rMIPPP_cond_loc</td>
<td>Generate a Marked Poisson point process (conditional on location)</td>
</tr>
<tr>
<td>rMIPPP_cond_mark</td>
<td>Generate a Marked Poisson point process (conditional on mark)</td>
</tr>
<tr>
<td>rGRF</td>
<td>Generate a Gaussian Random Field (used for MIPPPs with continuous marks)</td>
</tr>
<tr>
<td>MaternCov</td>
<td>Matern covariance function (used with rGRF)</td>
</tr>
</tbody>
</table>

Table 1: Simulations: these functions can be used to simulate important objects used to create and model IPPPs and MIPPPs.

### Mixture functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Task Accomplished</th>
</tr>
</thead>
<tbody>
<tr>
<td>approx_normmix</td>
<td>Approximate the masses of bivariate normal mixture components</td>
</tr>
<tr>
<td>dnormmix</td>
<td>Calculate the density or intensity of a normal mixture over a fine grid</td>
</tr>
<tr>
<td>normmix</td>
<td>Create a 2d mixture with normal components</td>
</tr>
<tr>
<td>to_int_surf</td>
<td>Convert a normal mixture to an intensity surface</td>
</tr>
</tbody>
</table>

Table 2: Mixture functions: these functions can be used to create and manipulate mixtures of normal components that can be later used to model the intensity surface of an IPPP.

### Model fit

<table>
<thead>
<tr>
<th>Model fit</th>
<th>Task Accomplished</th>
</tr>
</thead>
<tbody>
<tr>
<td>est_mix_damcmc</td>
<td>Estimate a mixture model parameters using DAMCMC</td>
</tr>
<tr>
<td>est_mix_bdmcmc</td>
<td>Estimate a mixture model parameters using BDMCMC</td>
</tr>
<tr>
<td>est_MIPPP_cond_mark</td>
<td>Fit a MIPPP conditionally on mark</td>
</tr>
<tr>
<td>est_MIPPP_cond_loc</td>
<td>Fit a MIPPP conditionally on location</td>
</tr>
<tr>
<td>est_intensity_np</td>
<td>Estimate the intensity surface using a non-parametric method</td>
</tr>
</tbody>
</table>

Table 3: Model Fitting: these functions can be used to fit an IPPP or MIPPP model.
<table>
<thead>
<tr>
<th>Function</th>
<th>Task Accomplished</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop_realization</td>
<td>Drop MCMC realizations</td>
</tr>
<tr>
<td>GetBDComplfit</td>
<td>Retrieve parts of a BDMCMC fit</td>
</tr>
<tr>
<td>GetBDTable</td>
<td>Retrieve the MAP and distribution of the number of components</td>
</tr>
<tr>
<td>GetBMA</td>
<td>Compute the Bayesian Model average surface</td>
</tr>
<tr>
<td>GetDensityValues</td>
<td>Retrieve density values</td>
</tr>
<tr>
<td>GetIPPPPLikValue</td>
<td>Retrieve the IPPP likelihood value</td>
</tr>
<tr>
<td>GetKLEst</td>
<td>Retrieve the surface of Kullback-Leibler (KL) estimators</td>
</tr>
<tr>
<td>GetMAPEst</td>
<td>Retrieve the surface of MAP estimators</td>
</tr>
<tr>
<td>GetMAPLabels</td>
<td>Retrieve the MAP estimates for the component labels</td>
</tr>
<tr>
<td>GetStats</td>
<td>Retrieves basic Bayesian estimates from a generated chain</td>
</tr>
<tr>
<td>GetPMEst</td>
<td>Retrieve the surface of Posterior Means</td>
</tr>
<tr>
<td>summary.bdmcmc_res</td>
<td>Summarize BDMCMC results</td>
</tr>
<tr>
<td>summary.damcmc_res</td>
<td>Summarize DAMCMC results</td>
</tr>
<tr>
<td>summary.intensity_surface</td>
<td>Summarize a 2d mixture with normal components</td>
</tr>
<tr>
<td>summary.MIPPP_fit</td>
<td>Summarize MIPPP results</td>
</tr>
<tr>
<td>summary.normmix</td>
<td>Summarize a 2d mixture with normal components</td>
</tr>
<tr>
<td>summary.sppmix</td>
<td>Summarize a point pattern from a Poisson process</td>
</tr>
</tbody>
</table>

Table 4: Processing MCMC output: these functions can be used to process a model fit output for an IPPP or MIPPP model. Note that the function call in the summary functions is simply 'summary'. What follows after the ‘.’ is just the class of the object and is not needed to obtain the summary.
<table>
<thead>
<tr>
<th>Function</th>
<th>Task Accomplished</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_labels</td>
<td>Check for label switching</td>
</tr>
<tr>
<td>CompareSurfs</td>
<td>Quantify the difference between two surfaces</td>
</tr>
<tr>
<td>FixLS_da</td>
<td>Fix label switching</td>
</tr>
<tr>
<td>Get_Rdiag</td>
<td>Reports the Gelman-Rubin convergence diagnostic</td>
</tr>
<tr>
<td>ktest2d</td>
<td>Nonparametric Goodness-of-fit test between two point patterns</td>
</tr>
<tr>
<td>ktest2dsurf</td>
<td>Nonparametric Goodness-of-fit test for a point pattern against a surface</td>
</tr>
<tr>
<td>mc_gof</td>
<td>Monte Carlo goodness of fit test</td>
</tr>
<tr>
<td>plot_autocorr</td>
<td>produces an autocorrelation plot to help assess convergence</td>
</tr>
<tr>
<td>plot_chains</td>
<td>Plot MCMC chains in order to visually inspect convergence</td>
</tr>
<tr>
<td>plot_CompDist</td>
<td>Plots for the number of components (histogram and MCMC chain)</td>
</tr>
<tr>
<td>plot_convdiags</td>
<td>produces both the autocorrelation and running means plots</td>
</tr>
<tr>
<td>plot_runmean</td>
<td>creates a running means plot in order to help assess convergence</td>
</tr>
<tr>
<td>selectMix</td>
<td>Mixture Model Selection for the intensity surface of an IPPP</td>
</tr>
</tbody>
</table>

Table 5: Model Diagnostics: these functions can be used to assess convergence of a model fit, perform goodness of fit tests, model selection and model comparison for an IPPP or MIPPP model.
<table>
<thead>
<tr>
<th>Function</th>
<th>Task Accomplished</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_title</td>
<td>Add a title to an existing ggplot2 plot</td>
</tr>
<tr>
<td>openwin_sppmix</td>
<td>Opens a new graphics window</td>
</tr>
<tr>
<td>plot.bdmcmc_res</td>
<td>Plot results from a BDMCMC fit</td>
</tr>
<tr>
<td>plot.damcmc_res</td>
<td>Plot results from a DAMCMC fit</td>
</tr>
<tr>
<td>plot.intensity_surface</td>
<td>Plots a normal mixture intensity in 3d</td>
</tr>
<tr>
<td>plot.MIPPP_fit</td>
<td>Fit a MIPPP conditionally on location</td>
</tr>
<tr>
<td>plot.normmix</td>
<td>Plot a mixture of normal components</td>
</tr>
<tr>
<td>plot.sppmix</td>
<td>Plot a spatial point pattern generated from a Poisson point process with mixture intensity surface</td>
</tr>
<tr>
<td>plot2dPP</td>
<td>Plot a spatial point pattern</td>
</tr>
<tr>
<td>plotmix_2d</td>
<td>2d exploratory plots for mixture intensity surfaces</td>
</tr>
<tr>
<td>plotmix_3d</td>
<td>Plot the density or intensity of a normal mixture in 3d over a fine grid</td>
</tr>
<tr>
<td>Plots_off</td>
<td>Closes all open plots</td>
</tr>
<tr>
<td>PlotUSAStates</td>
<td>Advanced visualization of USA states and their counties</td>
</tr>
<tr>
<td>plot_avgsurf</td>
<td>Plot the average intensity surface</td>
</tr>
<tr>
<td>plot_density</td>
<td>Plots a density or image</td>
</tr>
<tr>
<td>plot_ind</td>
<td>Plot membership indicators</td>
</tr>
<tr>
<td>plot_MPP_fields</td>
<td>Plot the mark probability fields</td>
</tr>
<tr>
<td>plot_MPP_probs</td>
<td>Plot the mark probabilities for each event of a marked point pattern</td>
</tr>
<tr>
<td>plot_true_labels</td>
<td>Plot the true membership indicators</td>
</tr>
<tr>
<td>Save_AllOpenRglGraphs</td>
<td>Saves all open RGL plots and closes the graphic devices</td>
</tr>
</tbody>
</table>

Table 6: Plotting: these functions are used to visualize created objects and model fits, visualize the USA states or counties and manipulate plots in general. Note that the function call in the plot functions is simply ‘plot’. What follows after the ‘.’ is just the class of the object and is not needed to produce the plot.
2 Fitting an IPPP using the sppmix package

In this section we illustrate via simulated examples how to simulate visualize and estimate an IPPP model using the sppmix package. We load the sppmix package first.

R code 1

library(sppmix)

2.1 Create the normmix and intensity surface objects

Using functions \texttt{rnormmix} and \texttt{to\_int\_surf} we generate a mixture with 4 normal components (a \texttt{normmix} object) and then use it to create an \texttt{intensity\_surface} object, representing the intensity surface of an IPPP. Alternatively, we can use function \texttt{rmixsurf} in order to generate the surface directly or function \texttt{normmix} to build a normal mixture with specific parameters and then build the surface using function \texttt{to\_int\_surf}. We choose the window $W = [-2, 2] \times [-2, 2]$ for this simulation so that we can have some of the mass of the components outside the window (i.e., introduce edge effects) and $\lambda = 500$ (the average number of points we expect to see). Note that we use the \texttt{summary} function after each call. The function calls are as follows:

R code 2

\texttt{truemix4=rnormmix(m = 4, sig0 = 2, df = 5, xlim = c(-2,2), ylim = c(-2,2))}

R code 3

\texttt{summary(truemix4)}

Output 1

Component 1 is centered at $[-0.77, -0.32]$ with probability 0.1098

\begin{verbatim}
covariance matrix:
   0.6197 -0.3561
-0.3561 1.0192
\end{verbatim}

Component 2 is centered at $[-0.18, -0.99]$ with probability 0.3942

\begin{verbatim}
covariance matrix:
   0.6638 0.0166
0.0166 1.2572
\end{verbatim}

Component 3 is centered at $[1.6, 1.41]$ with probability 0.2954

\begin{verbatim}
covariance matrix:
   0.4355 0.0241
0.0241 0.5590
\end{verbatim}

Component 4 is centered at $[1.67, -1.43]$ with probability 0.2006

\begin{verbatim}
covariance matrix:
\end{verbatim}
R code 4

```r
trueintsurfmix4=to_int_surf(truemix4, lambda = 500,
win=spatstat::owin(c(-2,2),c(-2,2)))
```

R code 5

```r
summary(trueintsurfmix4)
```

Output 2

Expected number of points over the window: 500
window: rectangle = [-2, 2] x [-2, 2] units
Normal Mixture with 4 component(s).
Component 1 is centered at [-0.77 , -0.32 ]
with probability 0.1098
  covariance matrix:
    0.6197 -0.3561
    -0.3561 1.0192
Component 2 is centered at [ -0.18 , -0.99 ]
with probability 0.3942
  covariance matrix:
    0.6638  0.0166
    0.0166  1.2572
Component 3 is centered at [ 1.6 , 1.41 ]
with probability 0.2954
  covariance matrix:
    0.4355  0.0241
    0.0241  0.5590
Component 4 is centered at [ 1.67 , -1.43 ]
with probability 0.2006
  covariance matrix:
    0.3495 -0.0622
    -0.0622 1.1832

2.2 Plotting the surfaces and visualizing edge effects

We plot the generated mixture density and intensity surface in 2d and 3d using the following function calls. We use the spatstat function `spatstat::owin` to build the two windows of interest.

R code 6

```r
origwin=spatstat::owin(c(-2,2),c(-2,2))
```
bigwin = spatstat::owin(c(-5,5), c(-5,5))
plot(truemix4, xlim = c(-2,2), ylim = c(-2,2),
title1 = "True mixture of normals density")
plot(trueintsurfmix4, main="True IPPP intensity surface")
plotmix_2d(trueintsurfmix4, win=origwin, colors = TRUE) +
  add_title("True IPPP intensity surface, W=[-2,2]x[-2,2]",
  lambda=trueintsurfmix4$lambda, m=trueintsurfmix4$m)

Figure 1 presents the 2d and 3d plots of the generated mixture density (top row) and intensity surface (bottom row) with 4 normal components. A call to function `approx_normmix` reveals the mass of each component within the given window. If all masses are 1 we do not have edge effects, otherwise, we need to set variable `truncate` to TRUE as needed in order to handle edge effects when we perform the statistical analysis.

**R code 7**

```r
approx_normmix(truemix4, xlim = c(-2,2), ylim = c(-2,2))
```

**Output 3**

0.8849419 0.7983823 0.5765053 0.5080802

Clearly, there are edge effects for $W_1 = [-2,2] \times [-2,2]$, which may lead to biased estimates of the parameters of the intensity surface. Increasing the window to $W_2 = [-5,5] \times [-5,5]$ we notice that edge effects are not present (the mass of each component is within the window). Next, we create the corresponding intensity surface object over the enlarged window and draw a point pattern using function `rsppmix` over the larger window (note the parameter `truncate=FALSE`).

**R code 8**

```r
approx_normmix(truemix4, xlim = c(-5,5), ylim = c(-5,5))
trueintsurfmix4bigwin = to_int_surf(trueintsurfmix4, win=bigwin)
ppmix4bigwin = rsppmix(intsurf = trueintsurfmix4bigwin, truncate = FALSE)
```

**Output 4**

0.9999981 0.9998243 0.9999991 0.9994885

A plot of the true allocation (membership) variables is obtained using function `plot_true_labels` and is displayed in figure 2.

**R code 9**

```r
plot_true_labels(ppmix4bigwin)
```
Plotting the density over different windows illustrates the problems encountered when working with point patterns in the presence of edge effects. More precisely, we plot the intensity over several windows in 2d using function `plotmix_2d` and note that depending on the choice of window we have more or fewer events. Here we used function `Count_pts` to count the number of points from the generated point pattern within the different windows and passed it to function `add_title` in order to display the correct number of points. Figure 3 contains 4 plots of the intensity surface over different windows.

**R code 10**

```r
plotmix_2d(trueintsurfmix4bigwin, ppmix4bigwin, win= spatstat:::owin(c(-1,1),c(-1,1)), colors = TRUE, truncate = FALSE)+
add_title("True IPPP intensity surface, W=[-1,1]x[-1,1]",
lambda=trueintsurfmix4bigwin$lambda, m=trueintsurfmix4bigwin$m, n=Count_pts(ppmix4bigwin, spatstat:::owin(c(-1,1),c(-1,1))))

plotmix_2d(trueintsurfmix4bigwin, ppmix4bigwin, win=origwin, colors = TRUE, truncate = FALSE)+
add_title("True IPPP intensity surface, W=[-2,2]x[-2,2]",
lambda=trueintsurfmix4bigwin$lambda, m=trueintsurfmix4bigwin$m, n=Count_pts(ppmix4bigwin, origwin))

plotmix_2d(trueintsurfmix4bigwin, ppmix4bigwin, win= spatstat:::owin(c(-3,3), c(-3,3)), colors = TRUE, truncate = FALSE)+
add_title("True IPPP intensity surface, W=[-3,3]x[-3,3]",
lambda=trueintsurfmix4bigwin$lambda, m=trueintsurfmix4bigwin$m, n=Count_pts(ppmix4bigwin, spatstat:::owin(c(-3,3),c(-3,3))))

plotmix_2d(trueintsurfmix4bigwin, ppmix4bigwin, win= spatstat:::owin(c(-4,4), c(-4,4)), colors = TRUE, truncate = FALSE)+
add_title("True IPPP intensity surface, W=[-4,4]x[-4,4]",
lambda=trueintsurfmix4bigwin$lambda, m=trueintsurfmix4bigwin$m, n=Count_pts(ppmix4bigwin, spatstat:::owin(c(-4,4),c(-4,4))))
```

Assuming that edge effects are not present, the window of observation has no effect in calculations and therefore, parameter truncate is set to FALSE in all functions called above. Note that the colors=TRUE setting will display the points with different colors depending on the true component the point was generated from. Finally, figure 4 presents the true surface over the enlarged window in 2d and 3d.
2.3 Mixture deconvolution and label switching

Label switching is a common issue in the Bayesian analysis of a mixture model, in particular, when the mixture components are not clearly separated from each other. When present it will severely affect the identifiability of the posterior distribution of parameters. More precisely, it does not allow for mixture deconvolution, that is, we can compute the average of the surfaces corresponding to posterior realizations of the \( p \)'s, \( \mu \)'s and \( \Sigma \)'s, or the Bayesian model average and propose it as an estimator of the true IPPP intensity.

However, this approach does not allow us to achieve mixture deconvolution, that is, we want to propose specific values for the parameters of the mixture model, including the exact number of components, as well as, the corresponding probabilities, means and covariance matrices. This can be crucial in some applications, e.g., when the component means represent the locations of monitoring stations, we need to explicitly estimate the values of the mixture component means. As a result, we can make decisions about where to allocate resources more efficiently. A natural choice for the number of components \( m \) that allows us to achieve mixture deconvolution is the MAP estimate of the distribution of the number of components, as obtained from a BDMCMC fit.

When label switching is present, computing the posterior means based on the MCMC will lead to inappropriate estimators of the parameters. As a result, if we want to achieve mixture deconvolution, then we have to fix the label switching problem (if present).

Obtaining estimates of the intensity surface that are not affected by label switching is a simple yet effective way of solving the problem. One such approach is to use an estimator for the mixture parameters that does not involve sums of the realizations. A straightforward approach is to use the MAP estimates of the parameters of the intensity surface. Using function \texttt{GetMAPEst} we obtain and display the MAP surface in the examples that follow.

Another approach to address the issue is to restrict the parameter space to clear up any ambiguity that might arise, for example, by specifying that the means of the mixture components are in a non-decreasing order. However, implementation of any such constraint would restrict the sampling scheme from exploring the entire parameter space resulting in a bias. This method is referred to as the Identifiability Constraint (IC) permutation of the component labels.

A more appropriate approach would be to conduct post-simulation processing. For example, we find the permutation of the parameters that minimizes some penalty function, and then apply that permutation to all the simulated realizations in order to compute the posterior mean estimators. The penalty or loss function we have implemented is the Squared Error Loss (SEL), and typically this approach is more reliable than the IC approach. Unfortunately, there is great computational cost if the number of components satisfies \( m \geq 7 \), since we have to go through \( m! \) permutations of each realization each time, and as \( m \) increases the number of iterations is impractical (extremely slow operation for a large number of posterior realizations).
The `sppmix` function `CheckLabels` can be used to assess if label switching is present, in which case, we can use function `FixLS_da` in order to permute the labels (and all of the posterior realizations). In both cases (IC or SEL permuted labels), Bayesian analysis proceeds as usual on the permuted realizations. In order to detect label switching, a simple visual inspection of the trace plots is usually enough in cases with severe label switching. In addition, and since we want to automate the process, we provide a heuristic method for detecting label switching in the function `CheckLabels`. More precisely, given some lag value, we check for sharp changes in the generated chains by computing the posterior means of a parameter (typically the x-y coordinates of a component mean), based on the past and future realizations around the current realization. In the function `FixLS_da` the lag value is small at the beginning and is increased gradually, until the function `CheckLabels` reports that there is no label switching present.

### 2.4 Fitting an IPPP via DAMCMC without edge effects

The function `est_mix_damcmc` can be used to fit the DAMCMC for a fixed number of mixture components. We call the routine with the truncate parameter set to FALSE (no edge effects) and entertain a mixture with \( m = 4 \) components (the true number). Note that we operate on window \( W_2 = [-5,5] \times [-5,5] \) for now. The result is a `damcmc_res` object representing the DAMCMC fit, to which we can apply the `summary` and `plot` functions.

```R
R code 12

DAMCMCfit=est_mix_damcmc(pp=ppmix4bigwin, m=4, L=50000, truncate=FALSE)
summary(DAMCMCfit)
plot(DAMCMCfit)
```

**Output 6**

```
-------------- Component 1 --------------
Probability: posterior mean = 0.272
95 % Credible Set:
[ 0.1800 , 0.3453 ]

Mean vector, x-coord: post mean = 1.575
95 % Credible Set:
[ 1.343 , 1.768 ]

Mean vector, y-coord: post mean = 1.365
95 % Credible Set:
[ 1.112 , 1.587 ]

Covariance, (1,1):
post mean = 0.43
95 % Credible Set:
[ 0.2966 , 0.6145 ]
```
Covariance, (1,2) and (2,1):
post mean = 0.08986
95 % Credible Set:
[ -0.04721 , 0.27209 ]

Covariance, (2,2):
post mean = 0.6257
95 % Credible Set:
[ 0.4022 , 0.9233 ]

---------------- Component 2 ----------------
Probability: posterior mean = 0.2582
95 % Credible Set:
[ 0.04736 , 0.51699 ]

Mean vector, x-coord: post mean = 0.1315
95 % Credible Set:
[ -0.7131 , 1.9218 ]

Mean vector, y-coord: post mean = -0.7626
95 % Credible Set:
[ -1.7433 , 0.4105 ]

Covariance, (1,1):
post mean = 0.694
95 % Credible Set:
[ 0.2903 , 1.4842 ]

Covariance, (1,2) and (2,1):
post mean = -0.0239
95 % Credible Set:
[ -0.6307 , 0.5666 ]

Covariance, (2,2):
post mean = 1.059
95 % Credible Set:
[ 0.345 , 2.025 ]

---------------- Component 3 ----------------
Probability: posterior mean = 0.2492
95 % Credible Set:
[ 0.06269 , 0.50107 ]

Mean vector, x-coord: post mean = 0.3418
95 % Credible Set:
[ -0.6699 , 1.9592 ]
Mean vector, y-coord: post mean = -0.8253
95 % Credible Set:
[ -1.7859 , 0.3728 ]

Covariance, (1,1):
post mean = 0.7204
95 % Credible Set:
[ 0.3011 , 1.5409 ]

Covariance, (1,2) and (2,1):
post mean = -0.0006067
95 % Credible Set:
[ -0.6325 , 0.5990 ]

Covariance, (2,2):
post mean = 1.071
95 % Credible Set:
[ 0.3607 , 2.1515 ]

---------------- Component 4 ----------------

Probability: posterior mean = 0.2205
95 % Credible Set:
[ 0.06146 , 0.46095 ]

Mean vector, x-coord: post mean = 0.6299
95 % Credible Set:
[ -0.6737 , 2.0129 ]

Mean vector, y-coord: post mean = -0.8465
95 % Credible Set:
[ -1.8061 , 0.3233 ]

Covariance, (1,1):
post mean = 0.7245
95 % Credible Set:
[ 0.2948 , 1.5431 ]

Covariance, (1,2) and (2,1):
post mean = 0.01041
95 % Credible Set:
[ -0.6523 , 0.6305 ]

Covariance, (2,2):
post mean = 1.086
95 % Credible Set:
[ 0.3779 , 2.3684 ]

Figure 5 illustrates typical plots of interest for a DAMCMC fit, including 2d and 3d plots of the surface of posterior means (top row), plot of the posterior means of the indicator variables and chains
for \( p \) (middle row), and trace plots for the component means (bottom row). The plots in the top row of figure 5 can be obtained from the DAMCMC fit object using first function \texttt{GetPMEst} to obtain the surface of posterior means and then plot it in 2d and 3d as we would do any surface object (e.g., see section 2.2 above, plots omitted). By default, a burn-in of 10\% of the total number of realizations is applied. Similarly, function \texttt{plot_ind} can be used to obtain the plot of the posterior means of the membership indicators.

### R code 13

```r
#just the code, plots omitted
post_mix = GetPMEst(DAMCMCfit)

plot(post_mix)

plot_ind(DAMCMCfit) + add_title("Posterior Means of the membership indicators", m = DAMCMCfit$m, n = DAMCMCfit$data$n)
```

We discuss briefly the results about the component probabilities \( p \). In particular, the posterior means are given by \([0.2720372, 0.2582063, 0.2492101, 0.2205464]\) with corresponding 95\% credible sets \([0.1800,0.3453], [0.04736,0.51699], [0.06269,0.50107] \) and \([0.06146,0.46095]\), respectively. Note that the posterior means are approximately 0.25, and recall that the true values are \([0.1097771, 0.394248, 0.2954111, 0.2005637]\). This is a first indication that there is label switching across the iterations of the DAMCMC. Further note that the index of each component has changed from what it was in the truth, which is not unusual since the DAMCMC (or BDMCMC) is run with starting values for the parameters selected randomly. As a result, the assignment of events to component 1 in the truth is not necessarily the same component based on a model fit (see figure 8 in order to fully appreciate how the indices may change).

#### 2.4.1 Identifying and fixing label switching

Looking at the trace plots it is obvious that the chains jump erratically from time to time, indicating that the labels have switched across iterations of the MCMC. Moreover, the clustering of the posterior means (marked by 'x') is yet another indication of label switching. Consequently, the posterior means of the mixture parameters and the corresponding surface of posterior means result in estimators that are biased. Note that the trace plots of figure 5 can be obtained also via a call to function \texttt{plot_chains} (plot omitted). Such plots can be particularly useful in identifying label switching. In addition, we can use function \texttt{check_labels} to assess heuristically the presence of label switching.

### R code 14

```r
#plot omitted
plot_chains(DAMCMCfit, separate = FALSE)

check_labels(DAMCMCfit)
```

### Output 7
Checking for label switching... Label switching present. Permute the labels to get a better fit, or obtain the average of the surfaces [1] TRUE

In order to fix the problem we compute the IC and SEL permuted realizations in function `FixLS_da`. The return value is a `damcmc_res` object with all the traits of a DAMCMC fit object with the only difference being that the realizations have been permuted. The function calls are as follows.

**R code 15**

```r
post_fixedIC = FixLS_da(DAMCMCfit)
plot(post_fixedIC)

post_fixedSEL = FixLS_da(DAMCMCfit, approx=FALSE)
plot(post_fixedSEL)
```

Figures 6 and 7 present similar plots to figure 5 but for the IC and SEL permuted realizations. We can clearly see an improvement in the posterior mean estimates in this case, with the SEL being much better than the IC (which is typically the case).

We take a closer look at the results from the SEL permuted fit for the component probabilities $p$. The posterior means are given by $[0.2713132, 0.3195929, 0.2116225, 0.1974714]$ with corresponding 95% credible sets $[0.1770, 0.3454]$, $[0.07162, 0.53626]$, $[0.03302, 0.46407]$ and $[0.07287, 0.42497]$, respectively. The indices for the SEL fit corresponding to the truth are as follows: component 1 corresponds to component 3 in the truth, 2 to 2, 3 to 1, and 4 to 4 (recall that the true values are $[0.1097771, 0.394248, 0.2954111, 0.2005637]$). Clearly, all of the true means are contained in the corresponding credible sets, however, the components are so close to each other that they cause some of the posterior means not to be exactly the corresponding true value. Results for the component probabilities and means are summarized in table 7. For example, the posterior mean for $\mu_1$ from the SEL fit is $\hat{\mu}_1 = [1.575210, 1.366443]$, and it corresponds to the true mean $\mu_3 = [1.601975, 1.405732]$. Using function `GetStats` we obtain the 95% credible sets for the x-y coordinates as $[1.342742, 1.766700]$ and $[1.112579, 1.587782]$, with both sets containing the true value. The code is as follows.

**R code 16**

```r
GetStats(post_fixedSEL$genmus[1,1,])$CredibleSet
GetStats(post_fixedSEL$genmus[1,2,])$CredibleSet
```

**Output 8**

```
[,1] 1.342742
[2,] 1.766700

[,1] 1.112579
[2,] 1.587782
```
Similarly for the remaining parameters, the true values are always within the 95% credible sets. For comparison purposes, figure 8 presents all the membership indicator variables starting with the true allocation variables, and the posterior means of the allocation variables based on the unpermuted labels, the IC permuted labels, and the SEL permuted labels.

**R code 17**

```r
plot_true_labels(ppmix4bigwin)

plot_ind(DAMCMCfit)+add_title("Posterior Means of the membership indicators (unpermuted labels)", m = DAMCMCfit$m, n = DAMCMCfit$data$n)

plot_ind(post_fixedIC)+add_title("Posterior means of the membership indicators (IC)", m=post_fixedIC$m, n=post_fixedIC$data$n)

plot_ind(post_fixedSEL)+add_title("Posterior means of the membership indicators (SEL)", m=post_fixedSEL$m,n = post_fixedSEL$data$n)
```

The following code summarizes how we can retrieve several posterior estimators of the intensity surface by appropriate estimation of the surface parameters, including PME (posterior mean estimator) and Kullback-Leibler estimator (KLE) of the mixture parameters, either for the unpermuted, IC permuted or SEL permuted posterior realizations.

**R code 18**

```r
SurfaceofPostMeans=GetPMEst(DAMCMCfit)
SurfaceofKLE=GetKLEst(DAMCMCfit)
permSurfaceofPostMeansIC=GetPMEst(post_fixedIC)
SurfaceofKLE_IC=GetKLEst(post_fixedIC)
permSurfaceofPostMeansSEL=GetPMEst(post_fixedSEL)
SurfaceofKLE_SEL=GetKLEst(post_fixedSEL)
```

We present 2d plots of all of these estimators in figure 9 with the left column displaying the PME and the right column the KLE of the intensity surface. Note that the KLE produces the same estimates as the PME for the component means and covariance matrices (for normal mixture components) but the probabilities are different. The function calls are as follows.

**R code 19**

```r
plotmix_2d(SurfaceofPostMeans, ppmix4bigwin,colors=TRUE,win=bigwin)+add_title(
```

22
All of these estimators can be used for mixture deconvolution, however, as per our discussion above, achieving mixture deconvolution is a hard problem in the presence of label switching, particularly, when the components are close to each other. There are ways to avoid the problem altogether as we see next.

2.4.2 Label switching free methods

We notice that even if we permute the realizations, the calculation of the MAP estimates of the parameters is not affected. Therefore, the MAP provides us with estimators of the mixture parameters (deconvolution) that are not affected by label switching, as well as, the corresponding surface of MAP estimators (MAPE) that is also invariant to label switching. Function `GetMAPEst` can be used to obtain these estimates and the call is as follows.

```R
R code 20
SurfaceofMAPs=GetMAPEst(DAMCMCfit)
plotmix_2d(SurfaceofMAPs, ppmix4bigwin,colors = TRUE, win=bigwin)+add_title("IPPP intensity surface of MAP estimates (unpermuted labels)", lambda =SurfaceofMAPs$lambda,m=SurfaceofMAPs$m,n=Count_pts(ppmix4bigwin,bigwin))
```
When mixture deconvolution is not of primary concern, then we can use function `plot_avgsurf` to obtain the average of the surfaces for each posterior realization (AoS) and avoid the label switching problem completely. However, computation of the average of the surfaces can be slow depending on the number of posterior realizations and how fine is the grid size (parameter LL affects the grid size of the function `plot_avgsurf`). Obviously, we expect a much smoother presentation of the surface for large LL. Note that the surface is also returned by this function so that it can be used for other plots. The following code shows how we can compute the average intensity and plot it in 2d and 3d using the `plot_density` and `plotmix_3d` functions, respectively.

```r
avgsurf=plot_avgsurf(DAMCMCfit,win=bigwin, LL = 100)

p=plot_density(as.data.frame(avgsurf))+ggplot2::ggtitle("Average surface of the posterior realization surfaces \nW=[-5,5]x[-5,5], x denotes a true mean")

pp_df=data.frame(ppmix4bigwin$x,ppmix4bigwin$y)
names(pp_df)=c("x", "y")

p=p+ggplot2::geom_point(data = pp_df,size=0.8)

mean_df=data.frame(do.call(rbind, trueintsurfmix4bigwin$mus))
names(mean_df)=c("x", "y")

print(p + ggplot2::geom_point(data = mean_df, color = "red",shape = "x", size = 5))

plotmix_3d(avgsurf, title1 = paste("Average surface of",.9*DAMCMCfit$L, "posterior surfaces"))
```

These plots along with the MAPE surface 2d and 3d plots can be seen in figure 10.

### 2.4.3 Comparing the surfaces

We can use function `CompareSurfs` to obtain measures that quantify the differences between two `intensity_surface` or `im` objects (this is done pixel wise), including the average squared pixel distance, the maximum absolute pixel distance, Frobenius norm of the matrix of pixel distances, average squared log pixel distance, and the Frobenius norm of the matrix of log pixel distances. Since we have the truth, we compute these measures for the true surface against all estimators we have seen up to this point. The following code computes the Frobenius norm of the matrix of log pixel distances and based on all comparison results it indicates the surface that gives the smallest value.
R code 22

```r
comp1=CompareSurfs(trueintsurfmix4, avgsurf, LL=100)
comp2=CompareSurfs(trueintsurfmix4, SurfaceofMAPs, LL=100)
comp3=CompareSurfs(trueintsurfmix4, SurfaceofKLE, LL=100)
comp4=CompareSurfs(trueintsurfmix4, SurfaceofPostMeans, LL=100)
comp5=CompareSurfs(trueintsurfmix4, permSurfaceofPostMeansIC, LL=100)
comp6=CompareSurfs(trueintsurfmix4, SurfaceofKLE_IC, LL=100)
comp7=CompareSurfs(trueintsurfmix4, permSurfaceofPostMeansSEL, LL=100)
comp8=CompareSurfs(trueintsurfmix4, SurfaceofKLE_SEL, LL=100)
ind=5
allFrobenius=c(comp1$dists[ind], comp2$dists[ind], comp3$dists[ind],
               comp4$dists[ind], comp5$dists[ind], comp6$dists[ind], comp7$dists[ind],
               comp8$dists[ind])
which.min(allFrobenius)
```

The values of the Frobenius norm are given by: AoS = 604.5522, MAPE = 1020.9866, KLE = 967.3645, PME = 962.0035, PME IC = 862.6409, KLE IC = 860.9392, PME SEL = 771.1940, and KLE SEL = 776.1199, and therefore the average of the surfaces across the posterior realizations emerges as the best estimate of the IPPP intensity.

2.4.4 Assessing convergence

Based on a `damcmc_res` object, we can use function `plot_convdiags` to produce many graphs to help assess convergence visually, including running mean plots and autocorrelation plots for all the parameters. This function calls `plot_runmean` and `plot_autocorr` for all parameters so we do not have to do it individually. In addition, we use function `Get_Rdiag` to obtain the Gelman-Rubin convergence diagnostic R (also known as the potential scale reduction), by producing k DAMCMC fits and compute the within-chain and between-chain variances. Values approximately equal to 1 indicate convergence, otherwise we need to run the chain for a longer number of iterations to get convergence.

The following code shows how we can assess convergence for the third component probability $p_3$. Note that since we have label switching using the unpermuted realizations the running mean plot does not indicate convergence (running mean is not almost a straight line), whereas, the IC and SEL permuted realizations indicate convergence. The plots are displayed in figure [11]. The Gelman-Rubin diagnostics are requested as well for all parameters and they give values near 1, however, note that we request to apply label switching algorithms using parameter `permute`, with 0 indicating unpermuted, the default value, 1 for IC permuted and 2 for SEL permuted realizations.
2.4.5 Model selection

One of the core variables of the mixture intensity surface is the number of components. As we will see in section 2.5, the BDMCMC can perform model selection with respect to the number of components automatically. However, when working with a fixed number of components we need to select the best \( m \) somehow. Therefore, we compute several classic model selection criteria using function `selectMix`, including Akaike’s Information Criterion (\( AIC \)), the Bayesian Information Criterion (\( BIC \)), and the Integrated Classification Likelihood Criterion (\( ICLC \)) defined by

\[
AIC = -2l(\hat{\lambda}, \hat{\theta}^{(m)}) + 2r,
\]

\[
BIC = -2l(\hat{\lambda}, \hat{\theta}^{(m)}) + r \log n,
\]

\[
ICLC = -2l(\hat{\lambda}, \hat{\theta}^{(m)}) + r \log n + 2En(\hat{z}_1, \ldots, \hat{z}_m),
\]

where \( l(.) \) denotes the log-likelihood, \( En(\hat{z}_1, \ldots, \hat{z}_m) = -\sum_{i=1}^m \sum_{j=1}^m \hat{z}_{ij} \log \hat{z}_{ij} \) denotes the estimated entropy of the classification variables, \( \hat{\lambda} \) and \( \hat{\theta}^{(m)} \) are estimators of the parameters \( \lambda \) and \( \theta^{(m)} = (p_{m-1}, \theta_1, \ldots, \theta_m) \) (e.g., the MLEs, MAPs or the posterior means), and \( r = 1 + (m-1) + m = 2m \), the number of parameters of the mixture model, and since \( \sum_{j=1}^m p_j = 1 \), where \( p_{m-1} = (p_1, \ldots, p_{m-1}) \). Models with small \( AIC \), \( BIC \)
and $ICLC$ values are preferred. The form of the log-likelihood for the $m$-component mixture intensity function is given by

$$l \left( \hat{\lambda}, \hat{\theta}^{(m)} \right) = -\log (n!) + n \log \hat{\lambda} - \hat{\lambda} + \sum_{i=1}^{n} \log \left( \sum_{j=1}^{m} \hat{p}_{j} \varphi_{j}(x_{i}|\hat{\theta}_{j}) \right),$$

where for large $n$, we approximate the term $n!$ using Stirling’s formula given by $n! \sim \sqrt{2\pi n}n^{n}e^{-n}$. The $ICLC$ is more appropriate to use for mixture models since it includes a term involving the entropy of the estimates of the indicator variables. See McLachlan and Peel (2000, chapter 6) for additional methods and details.

In order to avoid label switching issues that lead to inappropriate posterior means, we can approximate the mixture model density based on the posterior means $\sum_{j=1}^{m} \hat{p}_{j} \varphi_{j}(x_{i}|\hat{\theta}_{j})$ via the maximum of the mixture densities based on posterior realizations, namely, $\max_{k} \sum_{j=1}^{m} p_{j}^{(k)} \varphi_{j}(x_{i}|\theta_{j}^{(k)})$, where $p_{j}^{(k)}$ and $\theta_{j}^{(k)}$, $j = 1, 2, \ldots, m$, $k = 1, 2, \ldots, L$, are realizations from the posterior distribution $\pi(\theta^{(m)}|x_{1}, \ldots, x_{n})$. Similarly, we approximate $\sum_{j=1}^{m} \hat{z}_{ij} \log \hat{z}_{ij}$ with $\max_{k} \sum_{j=1}^{m} z_{ij}^{(k)} \log z_{ij}^{(k)}$, where the membership indicator $z_{ij}^{(k)}$ for the $i^{th}$ data point and the $j^{th}$ component at the $k^{th}$ iteration, is estimated by $p_{j}^{(k)} \varphi_{j}(x_{i}|\theta_{j}^{(k)}) / \sum_{r=1}^{m} p_{r}^{(k)} \varphi_{r}(x_{i}|\theta_{r}^{(k)})$.

From simulation studies we have observed that when label switching is present and the components are very close to each other this approximation works well. This case has been implemented in function `selectMix` by default (parameter runallperms=0). In other cases, we might want to run the label switching algorithm first and then compute the model selection criteria. This capability is included in the function by setting runallperms=1 (IC permuted) or runallperms=2 (SEL permuted), but the downside is computational cost as $m$ increases. The following call to function `selectMix` entertains mixture models for the IPPP intensity surface with 1 to 6 components and uses the approximation method discussed above (runallperms=0).

```
R code 24

ModelSel1=selectMix(ppmix4bigwin,1:6,truncate=FALSE)
```

The results are summarized in table 8. The AIC erroneously suggests 5 components, whereas, the BIC and ICLC correctly recommend a mixture with 4 components. Note that the function `selectMix` also invokes the BDMCMC in order to treat $m$ as random and obtain its distribution. Should the AIC, BIC and ICLC criteria disagree, we recommend that we use the BDMCMC in order to select the best number of components. See section 2.5 for more details on using the BDMCMC for estimating $m$ and the parameters of the mixture model for the IPPP intensity surface.

### 2.4.6 Model checking

We have included several functions that can perform goodness-of-fit (gof) tests, either to assess that a given point pattern arises from a specific IPPP model with mixture intensity, or that two point patterns arise from the same model. Function `mc_gof` performs a Monte Carlo test of gof for a given point pattern, where the entertained model is a Poisson with mixture of normals intensity surface. A nonparametric gof test between two point patterns is provided by function `kstest2d`. This function performs a two-dimensional Kolmogorov-Smirnov (K-S) gof test on two point patterns. Finally, function `kstest2dsurf`
can be used to perform a two-dimensional Kolmogorov-Smirnov gof test for a point pattern against a given intensity surface. This function essentially draws patterns from the entertained intensity surface many times, compares the given pattern against the drawn patterns using **kstest2d**, and then reports the percentage of time gof is appropriate. The code for these tests is given next.

### R code 25

```r
mc_gof(ppmix4bigwin, SurfaceofMAPs, 0.05)
mc_gof(ppmix4bigwin, SurfaceofPostMeans, 0.05)
mc_gof(ppmix4bigwin, SurfaceofKLE, 0.05)
mc_gof(ppmix4bigwin, permSurfaceofPostMeansIC, 0.05)
mc_gof(ppmix4bigwin, SurfaceofKLE_IC, 0.05)
mc_gof(ppmix4bigwin, permSurfaceofPostMeansSEL, 0.05)
mc_gof(ppmix4bigwin, SurfaceofKLE_SEL, 0.05)
```

All of these surfaces give a large p-value for the test. To see that not all models are appropriate we consider a surface with 2 components and rerun the test using the following code. We also draw a point pattern from this model and perform the MC gof test.

### R code 26

```r
mix2=normmix(ps=c(.4,.6), mus=list(c(-1,-1), c(2,2)),
sigmas = list(.1*diag(2), .1*diag(2)))
intsurfmix2=to_int_surf(mix2, lambda = 100, win = bigwin)
mc_gof(ppmix4bigwin, intsurfmix2, 0.05)
ppmix2=rsppmix(intsurf=intsurfmix2, truncate = FALSE)
mc_gof(ppmix2, intsurfmix2, 0.05)
```

The gof test yields a p-value of 0 in this case, indicating that the pattern does not arise from an IPPP with mixture intensity surface given by the object intsurfmix2. In contrast, testing the point pattern ppmix2 against the surface intsurfmix2 gives a p-value of 1, so that as expected, the intsurfmix2 surface provides gof for the point pattern ppmix2.

We also run the non-parametric tests using functions **kstest2d** and **kstest2dsurf** in order to compare the two patterns and the patterns against the surfaces described by the objects permSurfaceofPostMeansSEL and intsurfmix2. The code is as follows.

### R code 27

```r
kstest2d(ppmix2, ppmix4bigwin)
```
As expected, the point patterns ppmix4bigwin and ppmix2 arise from different models (the p-value for being from the same model is 2.2e-16). In addition, the K-S suggests gof for ppmix4bigwin against permSurfaceofPostMeansSEL 98.4% of the time for 500 K-S tests, but 0% gof against intsurfmix2, whereas, the K-S suggests gof for ppmix2 against intsurfmix2 99.6% of the time out of 500 K-S tests performed, and 0% gof against permSurfaceofPostMeansSEL.

2.5 Fitting an IPPP via BDMCMC without edge effects

We have seen several ways of selecting the number of components of the mixture intensity surface. When we assume that the number of components is random, \( m \) must be considered as a parameter of the model and it has its own distribution. The standard approach to Bayesian computation for the mixture parameters (e.g., DAMCMC) is not appropriate in this case since the parameter space can potentially change in dimension from iteration to iteration and therefore the standard Gibbs sampler cannot be applied. As a result, we require variable dimension samplers and we have implemented the BDMCMC by Stephens (2000) to help us in this context.

The function \texttt{est_mix_bdmcmc} can be used to fit the BDMCMC for a random number of mixture components. We call the routine with the truncate parameter set to FALSE (no edge effects) and entertain the maximum number of components to be \( m = 7 \) components, over the window \( W_2 = [-5, 5] \times [-5, 5] \). The result is a \texttt{bdmcmc_res} object representing the BDMCMC fit, to which we can apply the \texttt{summary} (output omitted) and \texttt{plot} functions.

\begin{verbatim}
R code 28
BDMCMCfit=est_mix_bdmcmc(pp = ppmix4bigwin, m = 7,L=100000,truncate = FALSE)
summary(BDMCMCfit)
plot(BDMCMCfit)
plot_CompDist(BDMCMCfit)
\end{verbatim}

The plots produced can be obtained any time using the same functions on the \texttt{bdmcmc_res} object as we have seen for a \texttt{damcmc_res}, and in addition, specific functions that handle BDMCMC output, such as \texttt{plot CompDist}. Figure 12 illustrates typical plots of interest from the BDMCMC fit, including 2d intensity and contour plots of the fitted intensity surface of posterior means corresponding to the MAP number of components \( m \), along with the distribution of the number of components and the generated chain for \( m \).

The BDMCMC produces a chain for the number of components that contains 16 realizations for 2 components, 2348 realizations for 3 components, 15571 realizations for 4 components, 32372 realizations...
for 5 components, 34413 realizations for 6 components, and 15280 realizations for 7 components. Not surprisingly, the algorithm never visits 1 components since it needs (at a minimum) at least 2 components to adequately model the point pattern. Each of these subsets of realizations can be extracted using function `GetBDCompfit` and they are essentially `damcmc_res` objects as we see below. Note that the MAP is erroneously reporting 5 components as the best choice, when the true number is 4. This problem is not present when the components are clearly separated from each other. We give some insight on this behavior of the BDMCMC fit next and how to fix the problem.

2.5.1 Dropping bad realizations

The BDMCMC can produce "bad" births or degenerate realizations (zero component mass) that end up in the chain for a specific number of components. In particular, since the algorithm either adds or removes one component at each realization, it is possible that bad components that will eventually die in subsequent iterations are still included as part of the chain for a specific value of components. This is not an issue of convergence, so that running the chain longer with a larger burn-in period will not solve this problem. Although these realizations do not affect averages of surfaces (no label switching problem and the corresponding component probability is zero), they have a detrimental effect when working for mixture deconvolution within the chain of a specific number of components, e.g., computing the posterior averages of the mixture parameters corresponding to the mixture with MAP number of components.

A `bdmcmc_res` object has a member `$Badgen` for each realization. If this parameter is 1 for some realizations, then dropping these degenerate realizations allows us to use the label switching algorithms efficiently and achieve mixture deconvolution. Obviously, the number of posterior realizations can drop significantly in number when we first apply burn-in and then drop the bad realizations, so it is good practice to run the BDMCMC for at least 20000 iterations.

The following code illustrates the problem and how we can handle it, especially in view of mixture deconvolution. First we drop 10% of the realizations (burn-in, use function `drop_realization` with defaults) and then we retrieve the frequency table and MAP estimate for number of components using function `GetBDTable`. We extract the realizations from the `bdmcmc_res` object using function `GetBDCompfit` and look at the range of the generated means (degenerate realizations are included). The member `$BDgens` contains these realizations.

### R code 29

```r
BDMCMCfitDropped=drop_realization(BDMCMCfit)
BDtab=GetBDTable(BDMCMCfit,FALSE)
MAPm=BDtab$MAPcomp
BDMCMCfitMAPcomp=GetBDCompfit(BDMCMCfit,MAPm)
BDMCMCfitMAPcompgens=BDMCMCfitMAPcomp$BDgens
range(BDMCMCfitMAPcompgens$genmus[,1,])
range(BDMCMCfitMAPcompgens$genmus[,2,])
```
We notice that the range of values for the x-coordinates of the means is -142.4059 to 130.6866 and for the y-coordinates -188.1274 to 254.3519. Such realizations have near zero component probability so that they do not affect the average of surfaces. However, they do lead to inappropriate estimates for the component means when we try to perform mixture deconvolution, e.g., the surface of averages.

We illustrate how to drop the bad realizations and fix the problem so that we can retrieve correct posterior mean estimates.

```r
R code 30

BDMCMCfitGood = drop_realization(BDMCMCfitDropped, (BDMCMCfitDropped$Badgen==1))

BDtabGood = GetBDTable(BDMCMCfitGood, FALSE)

MAPmGood = BDtabGood$MAPcomp

BDMCMCfitMAPcompGood = GetBDCompfit(BDMCMCfitGood, MAPmGood)

BDMCMCfitMAPcompgensGood = BDMCMCfitMAPcompGood$BDgens

range(BDMCMCfitMAPcompgensGood$genmus[,1,])
range(BDMCMCfitMAPcompgensGood$genmus[,2,])
```

As anticipated, the range of generated means is now -4.997066 to 4.867724 for the x-coordinate and -4.951227 to 4.534312 for the y-coordinate, so that all realizations of the means are within the observation window $W_2 = [-5, 5] \times [-5, 5]$.

### 2.5.2 Using the realizations corresponding to the MAP components

Now we check how many realizations are left and produce the corresponding surface of posterior means; we set the burnin parameter to 0 in these function calls since burn-in has been applied already. Note that the good realizations provide a MAP estimate for $m$ that correctly identifies the true number of components, namely 4. In particular, the chain for the number of components now contains 16 realizations for 2 components, 2163 realizations for 3 components, 9795 realizations for 4 components, 6439 realizations for 5 components, 2494 realizations for 6 components and 42 realizations for 7 components.

```r
R code 31

plot_CompDist(BDMCMCfitGood)

plot_ind(BDMCMCfitMAPcompgensGood, burnin=0)+add_title("Posterior means of the membership indicators (unpermuted)”,
m=BDMCMCfitMAPcompgensGood$m,n=BDMCMCfitMAPcompgensGood$data$n)

plotmix_2d(BDMCMCfitMAPcompGood$BDsurf, ppmix4bigrwin, win=bigrwin)+add_title("IPPP intensity surface of posterior means (MAP number of components)”,
lambda=BDMCMCfitMAPcompGood$BDsurf$lambda,m=BDMCMCfitMAPcompGood$BDsurf$m,
```
Furthermore, function `check_labels` indicates that label switching is present and therefore we permute the labels using the IC and SEL methods and compute the surfaces of posterior means.

**R code 32**

```r
post_fixedBDMCMCfitIC = FixLS_da(BDMCMCfitMAPcompgensGood,burnin=0)

permSurfaceofPostMeansBDIC=GetPMEst(post_fixedBDMCMCfitIC,burnin=0)

SurfaceofBDKLE_IC=GetKLEst(post_fixedBDMCMCfitIC,burnin=0)

plotmix_2d(permSurfaceofPostMeansBDIC,ppmix4bigwin,win=bigwin)+add_title("Poisson surface of posterior means (IC)",lambda=permSurfaceofPostMeansBDIC$lambda, m=permSurfaceofPostMeansBDIC$m,n=ppmix4bigwin$n,L=post_fixedBDMCMCfitIC$L)

plot_chains(post_fixedBDMCMCfitIC,separate = FALSE,burnin=0)

plot_ind(post_fixedBDMCMCfitIC,burnin=0)+add_title("Posterior means of the membership indicators (IC)", m=post_fixedBDMCMCfitIC$m, n = post_fixedBDMCMCfitIC$data$n)
```

**R code 33**

```r
post_fixedBDMCMCfitSEL = FixLS_da(BDMCMCfitMAPcompgensGood,approx=FALSE,burnin=0)

permSurfaceofPostMeansBDSEL=GetPMEst(post_fixedBDMCMCfitSEL,burnin=0)

SurfaceofBDKLE_SEL=GetKLEst(post_fixedBDMCMCfitSEL,burnin=0)

plotmix_2d(permSurfaceofPostMeansBDSEL,ppmix4bigwin,win=bigwin)+add_title("Poisson surface of posterior means (SEL)", lambda=permSurfaceofPostMeansBDSEL$lambda, m=permSurfaceofPostMeansBDSEL$m,n=ppmix4bigwin$n,L=post_fixedBDMCMCfitSEL$L)

plot(permSurfaceofPostMeansBDSEL,main="Poisson surface of posterior means (SEL)")

plot_chains(post_fixedBDMCMCfitSEL,separate = FALSE,burnin=0)

plot_ind(post_fixedBDMCMCfitSEL,burnin=0)+add_title("Posterior means of the membership indicators (SEL)", m=post_fixedBDMCMCfitSEL$m, n = post_fixedBDMCMCfitSEL$data$n)
```
The results of these code snippets are summarized in figures 13, 14, and 15 for the unpermuted realizations, the IC permuted realizations and the SEL permuted realizations, respectively. The top-left plot presents the distribution of \( m \) on all these figures. The top-right plots illustrate the surface of posterior means based on the 9795 realizations corresponding to the MAP=4 number of components, whereas, the middle-left plots present the estimated allocation variables. The trace plots for the component probabilities and the x-y coordinates of the posterior means are also presented. Clearly, the SEL outperforms the unpermuted and the IC permuted fit.

The following code can be used to obtain the label switching free estimates (MAPE and AoS) based on the realizations corresponding to MAP components.

R code 34

```
R code 34
```

avgsurfBDMAP=plot_avgsurf(BDMCMCfitMAPcompgens,win=bigwin,LL=100,burnin=0)

SurfaceofMAPs4BD=GetMAPEst(BDMCMCfitMAPcompgensGood)

2.5.3 The Bayesian Model Average

We anticipate that one of the best estimators of the true IPPP intensity surface will be the Bayesian model average (BMA) of all realizations (after burnin). We use function GetBMA in order to obtain and plot the BMA surface. The output from GetBMA is an im object that can be used with functions plot_density and plotmix_3d to produce the 2d and 3d plots. Note that calculation of the BMA is slow for a large number of realizations and number of components.

R code 35

```
R code 35
```

BDMCMCfit_BMA=GetBMA(BDMCMCfitDropped,win=bigwin)

p<-plot_density(BDMCMCfit_BMA)+ggplot2::ggtitle("Bayesian model average intensity surface Window=[-5,5]x[-5,5], x denotes a true mean")

pp_df <- data.frame(ppmix4bigwin$x,ppmix4bigwin$y)

names(pp_df) <- c("x", "y")

p<-p + ggplot2::geom_point(data = pp_df,size=0.8)

mean_df <- data.frame(do.call(rbind, trueintsurfmix4bigwin$mus))

names(mean_df) <- c("x", "y")

print(p + ggplot2::geom_point(data = mean_df, color = "red",shape = "x", size = 5))

title1=paste("Bayesian model average of",BDMCMCfitDropped$L, "posterior realizations")

plotmix_3d(BDMCMCfit_BMA,title1=title1)
Although the BMA intensity surface is a great estimator of the true IPPP intensity, it does not allow us to achieve mixture deconvolution. Next we collect all 2d and 3d plots of the best estimates we have seen from the BDMCMC for comparison purposes.

### 2.6 Comparing the DAMCMC and BDMCMC surfaces

As we saw in section [2.4.3](#), the average of the surfaces and the MAPE from the label switching free methods, and the PME and KLE for the SEL permuted realizations emerged as the best (in terms of small Frobenius norm) methods in estimating the true IPPP intensity surface. The first 4 comparisons of these surfaces against the truth are recomputed (for completeness) in the code below. In addition, we use the estimates based on the good realizations of the BDMCMC fit that correspond to the MAP=4, including the average of the surfaces, the PME and MAPE for the unpermuted, IC permuted and SEL permuted, and the BMA. Table [9](#) summarizes these results.

### R code 36

```
code1=CompareSurfs(trueintsurfmix4,avgsurf,LL=100)
code2=CompareSurfs(trueintsurfmix4,SurfaceofMAPs,LL=100)
code3=CompareSurfs(trueintsurfmix4,permSurfaceofPostMeansSEL,LL=100)
code4=CompareSurfs(trueintsurfmix4,SurfaceofKLE_SEL,LL=100)
code5=CompareSurfs(trueintsurfmix4,avgsurfBDMAP,LL=100)
code6=CompareSurfs(trueintsurfmix4,BDMCMCfitMAPcompGood$BDsurf,LL=100)
code7=CompareSurfs(trueintsurfmix4,SurfaceofMAPs4BD,LL=100)
code8=CompareSurfs(trueintsurfmix4,permSurfaceofPostMeansBDIC,LL=100)
code9=CompareSurfs(trueintsurfmix4,SurfaceofBDKLE_IC,LL=100)
code10=CompareSurfs(trueintsurfmix4,permSurfaceofPostMeansBDSEL,LL=100)
code11=CompareSurfs(trueintsurfmix4,SurfaceofBDKLE_SEL,LL=100)
code12=CompareSurfs(trueintsurfmix4,BDMCMCfit_BMA,LL=100)

ind=5
allFrobeniusDABD=c(compare1$dists[ind],compare2$dists[ind],
compare3$dists[ind],compare4$dists[ind],compare5$dists[ind],
compare6$dists[ind],compare7$dists[ind],compare8$dists[ind],
compare9$dists[ind],compare10$dists[ind],compare11$dists[ind],
compare12$dists[ind])
```
The AoS from the DAMCMC fit, the AoS from the BDMCMC fit for MAP=4 components and the BMA emerge as the best estimates of the true IPPP surface, with the second method being best among these label switching free methods. Regarding mixture deconvolution, the PME and KLE based on SEL permuted realizations from the DAMCMC and BDMCMC emerge as the best estimates, with the PME from the SEL permuted realizations of the DAMCMC fit being the best. Note that if we repeat all this analysis for a different IPPP we expect these surfaces to perform best especially in the presence of label switching.

The following code can be used to plot the aforementioned surfaces from the BDMCMC fit.

```r
R code 37

p <- plot_density(avgsurfBDMAP) + ggplot2::ggtitle("AoS for MAP=4 components
Window=[-5,5]x[-5,5], x denotes a true mean")

pp_df <- data.frame(ppmix4bigwin$x, ppmix4bigwin$y)
names(pp_df) <- c("x", "y")

p <- p + ggplot2::geom_point(data = pp_df, size=0.8)
mean_df <- data.frame(do.call(rbind, trueintsurfmix4bigwin$mus))
names(mean_df) <- c("x", "y")

print(p + ggplot2::geom_point(data = mean_df, color = "red", shape = "x", size=5))

plotmix_3d(avgsurfBDMAP, title1 = paste("Average surface of",
BDMCMCfitMAPcompgensGood$L,"posterior surfaces (BDMCMC)"))

plotmix_2d(SurfaceofMAPs4BD, ppmix4bigwin, colors = TRUE, win=bigwin) +
add_title("MAPE IPPP intensity surface for MAP=4 components, unpermuted labels")
lambda = SurfaceofMAPs4BD$lambda, m = SurfaceofMAPs4BD$m,
N = Count_pts(ppmix4bigwin, bigwin)

plot(SurfaceofMAPs4BD, main="MAPE IPPP intensity surface (MAP components)",
win=bigwin)

print(plotmix_2d(permSurfaceofPostMeansBDSEL, ppmix4bigwin, win=bigwin) +
add_title("Poisson surface of posterior means (SEL)"
lambda = permSurfaceofPostMeansBDSEL$lambda,
m = permSurfaceofPostMeansBDSEL$m, n = ppmix4bigwin$n,
L = post_fixedBDMCMCfitSEL$L))

35
We display all of these surfaces in 2d and 3d (including the MAPE and BMA) in figures 16 and 17 respectively.

### 2.7 Handling edge effects

Edge effects are major problem when working with point patterns, in particular, for non-stationary point processes such as the IPPP. Even if we define the point process over the whole \( \mathbb{R}^2 \), we always view it over a fixed window \( W \subset \mathbb{R}^2 \), since a point pattern observed from the IPPP is typically over \( W \). Therefore, any estimates of the parameters used to describe the intensity surface of the IPPP depend on \( W \) and in most cases they are not robust to the choice of window. This has led researchers to the creation of edge-corrected non-parametric estimators of the intensity function as well as parametric estimators of the parameters of the intensity function. See Cressie (1993), Illian et al. (2008), Diggle (2013) and Baddeley et al. (2015), for additional details.

We are taking care of edge effects by normalizing the mixture components so that the whole of their mass is within the window, and then obtain the estimates of mixture intensity surface parameters. In this way, we alleviate the effect of the window in our modeling approach of the intensity surface of the IPPP. We will treat the simulated example as an example of a point pattern with edge effects by looking at the process in the reduced window \( W_1 = [-2, 2] \times [-2, 2] \). We will not repeat all of the analyses of the previous sections in this section since the only difference in the function calls is simply a parameter `truncate`. In what we have seen up to this point, variable `truncate` was set to FALSE throughout (default value everywhere except for the function `rsppmix`). Consequently, in order to handle edge effects all we need to do is rerun the code snippets of the previous sections with variable `truncate` set to TRUE. We summarize the function calls in the following code.

```r
# for comparison, fix the window of the sampled pattern to be the small one # and use the point pattern we have already worked with

ppmix4origwin=ppmix4bigwin
ppmix4origwin$window=origwin

DAMCMCfitEE=est_mix_damcmc(pp=ppmix4origwin,m=4,L=100000,truncate=TRUE)
check_labels(DAMCMCfitEE)

BDMCMCfitEE=est_mix_bdmcmc(pp=ppmix4origwin,m=7,L=100000,truncate=TRUE)
BDMCMCfitDroppedEE=drop_realization(BDMCMCfitEE)
```

36
BDMCMCfitGoodEE = drop_realization(BDMCMCfitDroppedEE,
(BDMCMCfitDroppedEE$Badgen == 1))

Note that 139 events out of 503 events are outside the window $W_1$, so that we are missing a lot of the simulated data. This loss of information certainly affects the estimate of the IPPP surface. The following code collects the usual estimates of the IPPP intensity surface.

**R code 39**

```r
AoS_EE = plot_avgsurf(DAMCMCfitEE, win=origwin, LL=100)
post_fixedDAMCMCfitSEL = FixLS_da(DAMCMCfitEE, approx=FALSE)
PME_EE_SEL = GetPMEst(post_fixedDAMCMCfitSEL)
KLE_EE_SEL = GetKLEst(post_fixedDAMCMCfitSEL)
SurfaceofMAPSEe = GetMAPEst(DAMCMCfitEE, truncate = TRUE)
BDMCMCfit_BMA = GetBMA(BDMCMCfitGoodEE, win=origwin)
```

We display all of these surfaces in 2d and 3d in figures 18 and 19 respectively. The code that generates these plots is as follows. Clearly, once again we have indications that the BMA emerges as the best estimate.

**R code 40**

```r
plotmix_2d(trueintsurfmix4, ppmix4origwin, win=origwin, colors = TRUE) +
  add_title("True IPPP intensity surface, W=\([-2,2]\times\[-2,2]\)",
            lambda=trueintsurfmix4$lambda, m=trueintsurfmix4$m)

p = plot_density(AoS_EE) + ggtitle("AoS intensity surface
Window=\([-2,2]\times\[-2,2]\), x denotes a true mean")

pp_df = data.frame(ppmix4origwin$x, ppmix4origwin$y)

names(pp_df) = c("x", "y")

p = p + ggplot2::geom_point(data = pp_df, size = 0.8)

mean_df = data.frame(do.call(rbind, trueintsurfmix4$mus))

names(mean_df) = c("x", "y")

print(p + ggplot2::geom_point(data = mean_df, color = "black", shape = "x",
size = 5))
```
plotmix_3d(AoS_EE, title1=paste("Average surface of",.9*DAMCMCfitEE$L, 
"posterior surfaces (DAMCMC Edge-Corrected)"))

plotmix_2d(PME_EE_SEL, ppmix4origwin, colors = TRUE, win=origwin) +
add_title("PME IPPP intensity surface (Edge-Corrected, SEL permuted labels)",
lambda=PME_EE_SEL$lambda,m=PME_EE_SEL$m,
n=Count_pts(ppmix4origwin,origwin))

plot(PME_EE_SEL, main="PME IPPP intensity surface (Edge-Corrected, SEL)",
win=origwin)

plotmix_2d(KLE_EE_SEL, ppmix4origwin, colors=TRUE, win=origwin)+
add_title("KLE IPPP intensity surface (Edge-Corrected, SEL permuted labels)",
lambda=KLE_EE_SEL$lambda,m=KLE_EE_SEL$m,
n=Count_pts(ppmix4origwin,origwin))

plot(KLE_EE_SEL, main="KLE IPPP intensity surface (Edge-Corrected, SEL)",
win=origwin)

plotmix_2d(SurfaceofMAPsEE, ppmix4origwin, colors=TRUE, win=origwin)+
add_title("IPPP intensity surface of MAP estimates (Edge-Corrected)",
lambda=SurfaceofMAPsEE$lambda,m=SurfaceofMAPsEE$m,
n=Count_pts(ppmix4origwin,origwin))

plot(SurfaceofMAPsEE, main="MAPE IPPP intensity surface (Edge-Corrected)",
win=origwin)

p=plot_density(BDMCMCEEfit_BMA)+ggtitle( 
"BMA intensity surface (edge-corrected) 
Window=[-2,2]x[-2,2], x denotes a true mean")

pp_df=data.frame(ppmix4origwin$x,ppmix4origwin$y)
names(pp_df)=c("x", "y")

p=p+ggplot2::geom_point(data=pp_df, size=0.8)

mean_df=data.frame(do.call(rbind, trueintsurfmix4$mus))
names(mean_df)=c("x", "y")

print(p + ggplot2::geom_point(data = mean_df, color = "black",shape = "x",
size = 5))

plotmix_3d(BDMCMCEEfit_BMA,title1=
"Bayesian model average intensity surface (BDMCMC Edge-Corrected)")

38
In order to find the surface that gives the best fit we compute the comparison measures once again in the following code.

<table>
<thead>
<tr>
<th>R code 41</th>
</tr>
</thead>
<tbody>
<tr>
<td>compEE1=CompareSurfs(trueintsurfmix4,AoS_EE,LL=100)</td>
</tr>
<tr>
<td>compEE2=CompareSurfs(trueintsurfmix4,PME_EE_SEL,LL=100)</td>
</tr>
<tr>
<td>compEE3=CompareSurfs(trueintsurfmix4,KLE_EE_SEL,LL=100)</td>
</tr>
<tr>
<td>compEE4=CompareSurfs(trueintsurfmix4,SurfaceofMAPsEE,LL=100)</td>
</tr>
<tr>
<td>compEE5=CompareSurfs(trueintsurfmix4,BDMCMCEEfit_BMA,LL=100)</td>
</tr>
<tr>
<td>ind=5</td>
</tr>
<tr>
<td>allFrobeniusEE=c(compEE1$dists[ind], compEE2$dists[ind], compEE3$dists[ind], compEE4$dists[ind], compEE5$dists[ind])</td>
</tr>
<tr>
<td>which.min(allFrobeniusEE)</td>
</tr>
<tr>
<td>allFrobeniusEE</td>
</tr>
</tbody>
</table>

The Frobenius norms for the AoS, PME, KLE, MAPE and BMA against the true surface are given by 62.25771, 76.34342, 78.56762, 64.43584 and 58.24204, respectively. Therefore, the BMA is once again providing the best fit.

### 2.8 Using the BMA to find the best deconvolution surface

As good as the BMA or the AoS can be in approximating the true IPPP surface, they cannot be used for mixture deconvolution. From countless simulations we have ran, the BMA or the AoS are always better than any of the other estimates of the IPPP surface we can obtain, with or without edge effects present. To achieve mixture deconvolution (specially when we don’t have the truth to compare against, like in real life applications), we can compare the PME, KLE or MAPE surfaces against the BMA (or the AoS) and choose the best one. The following code calculates the distances against the BMA (results against the AoS omitted).

<table>
<thead>
<tr>
<th>R code 42</th>
</tr>
</thead>
<tbody>
<tr>
<td>compBMAEE1=CompareSurfs(BDMCMCEEfit_BMA,SurfaceofMAPsEE,LL=100)</td>
</tr>
<tr>
<td>compBMAEE2=CompareSurfs(BDMCMCEEfit_BMA,PME_EE_SEL,LL=100)</td>
</tr>
<tr>
<td>compBMAEE3=CompareSurfs(BDMCMCEEfit_BMA,KLE_EE_SEL,LL=100)</td>
</tr>
<tr>
<td>ind=5</td>
</tr>
</tbody>
</table>
The Frobenius norms for the MAPE, PME, and KLE against the BMA surface are given by 35.25432, 32.96793, 39.30651, respectively, and therefore, the PME is the best deconvoluted estimator of the intensity surface in this simulated example.

3 Fitting a MIPPP using the sppmix package

Next we present the functions we have implemented in the sppmix package and illustrate their use in order to accomplish modeling of a MIPPP (see paper section 3). As in the previous section, we consider only simulated examples for these illustrations.

3.1 Conditioning on marks

We use function `rMIPPP_cond_mark` in order to create a marked point pattern using randomization for the ground processes corresponding to $M = 4$ discrete marks. The mark distribution over $\xi = 1, 2, 3, 4$, is discrete with probabilities $\theta_2 = [.2, .5, .2, .1]$. We use a smaller window $W_1 = [-3, 3] \times [-3, 3]$ in order to generate the components for each ground process, but then we will use a larger window $W_2 = [-10, 10] \times [-10, 10]$ for better presentation. Therefore, in this example we assume that there are no edge effects so that we set parameter truncate to FALSE as needed. Note however that the analysis can be repeated with variable truncate set to TRUE as needed if the user needs to work with edge effects. The function call is as follows and it includes a call to function `Count_pts` so that we can make sure that no point is outside the big window. Using the `print` function on the object `newMPP` can provide additional information about the generated MIPPP (output omitted).

R code 43

```r
newMPP=rMIPPP_cond_mark(params=c(.2,.5,.2,.1),
bigwin=spatstat::owin(c(-10,10),c(-10,10)),truncate = FALSE)
print(newMPP)
bigwin=spatstat::owin(c(-10,10),c(-10,10))
Count_pts(newMPP$genMPP,bigwin)
plot2dPP(newMPP$genMPP)
```

The resulting object named `newMPP` is of class `MIPPP_fit`. It is a list containing a plethora of information about the created MIPPP conditional on marks, including: `$groundsurfs`, a list of `intensity_surface` objects containing the surfaces of the ground processes (one for each discrete mark value), `$groundPPs`, a list of `ppp` objects containing the locations of the ground processes (one for each discrete mark value), and `$genMPP`, the generated point pattern as an object of class `ppp`
and `sppmix`, where the member `$marks` contains the marks at each of the generated locations. The function `rMIPPP_cond_mark` also produces informative plots about the generated object. These plots are illustrated in figure [20]. In the top-left plot we display the MIPPP with the circles denoting the different marks. The next 4 plots show the true ground processes along with the events corresponding to each mark value. The last plot is the result of a call to function `plot2dPP` and it illustrates the unmarked events as a general point pattern.

Next we describe the steps needed to analyze this MIPPP. First we store the true number of components for each ground surface in a vector `m`. In general, we would not know these values of hand, but we can estimate the number of components with the methods we have seen in the previous section, including model selection criteria or running the BDMCMC for the point pattern corresponding to each mark value and choose the number of components for each ground process. Passing an integer `m` makes the function fit the BDMCMC and retrieve the BMA.

Now let us estimate the mark distribution parameters and the ground intensity parameters at each mark using function `est_MIPPP_cond_mark`. This function also checks for label switching automatically and permutes the labels if required.

### R code 44

```r
m = c(newMPP$groundsurfs[[1]]$m, newMPP$groundsurfs[[2]]$m, 
newMPP$groundsurfs[[3]]$m, newMPP$groundsurfs[[4]]$m)

MIPPPfit = est_MIPPP_cond_mark(newMPP$genMPP, m = m, truncate = FALSE)

summary(MIPPPfit)
plot(MIPPPfit)

MIPPPfitBMA = est_MIPPP_cond_mark(newMPP$genMPP, m = 6, truncate = FALSE)

summary(MIPPPfitBMA)
plot(MIPPPfitBMA)

# posterior analysis for the mark distribution
MIPPPfit$mark_dist
GetStats(MIPPPfit$gen_mark_ps[,1])$CredibleSet
GetStats(MIPPPfit$gen_mark_ps[,2])$CredibleSet
GetStats(MIPPPfit$gen_mark_ps[,3])$CredibleSet
GetStats(MIPPPfit$gen_mark_ps[,4])$CredibleSet
```

Generic methods such as `summary` and `plot` can be applied to the `MIPPP_fit` object returned from the `est_MIPPP_cond_mark` function calls (results omitted). The posterior means of the mark
probabilities are given by $[0.18080050, 0.52157065, 0.20109919, 0.09652965]$, and they are contained in the $mark\_dist$ member of the MIPPP\_fit object MIPPPfit. We also obtain 95% credible sets for the probability of observing each mark as: $[0.1481800, 0.2160714]$, $[0.4776688, 0.5656352]$, $[0.1667173, 0.2376189]$, and $[0.07200498, 0.12440242]$. Note that the true values $[2, 5, 2, 1]$ are included in the credible sets. Similar results are obtained using the MIPPPfitBMA fits.

We summarize the BDMCMC fits of the ground processes in figure 21. Any of the analyses presented in previous sections can be applied to the MIPPP\_fit objects MIPPPfit and MIPPPfitBMA and their elements, e.g., model selection and checking for a ground process corresponding to a specific mark value (results omitted).

### 3.2 Conditioning on locations

Using function rMIPPP\_cond\_loc we can simulate a MIPPP either for discrete or continuous marks. However, Bayesian estimation of the model parameters is implemented only for the discrete mark case (in function est\_MIPPP\_cond\_loc). If the marks are discrete valued, we use the aforementioned mark distribution (see paper section 3.2) and generate the mark values at the observed points, as well as, produce the probability fields over the window of observation. In the continuous mark case, we generate Gaussian random fields (or their transformations) in order to build the mark probability fields.

Note that the parameters $\gamma$ act as weights in the discrete mark distribution, where large values lead to small probabilities of observing the corresponding mark value. Function rMIPPP\_cond\_loc returns both the ground process and the mark distribution, as well as, plots the probability fields of observing a mark. The neighborhood system is controlled by the parameter $r$, with large values leading to smoother fields (since we borrow strength across space) whereas small values tend to concentrate the mark masses about the observed locations. For this example we illustrate a MIPPP with 3 discrete marks and modeled conditionally on location. We take $\gamma = [1.2, 1.5, 1.3]$ and $r = .5$ for the neighborhood system, and use the a small window $W_i = [-2, 2] \times [-2, 2]$, wherein we generate the mixture components. Moreover, this function allows us to enlarge the window (for better visualization) via parameter bigwin to $W_i = [-5, 5] \times [-5, 5]$. We also fit the Bayesian model for the MIPPP using function est\_MIPPP\_cond\_loc (returns an MIPPP\_fit object) and obtain posterior realizations of the model parameters, including $\gamma$ and the component mixture parameters for the ground process. The code is as follows.

<table>
<thead>
<tr>
<th>R code 45</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPcondloc=rMIPPP_cond_loc(gammas=c(1.2,1.5,1.3),r=.5,win=spatstat::owin(c(-2,2),c(-2,2)),bigwin=spatstat::owin(c(-5,5),c(-5,5)),L=10000)</td>
</tr>
<tr>
<td>L=50000</td>
</tr>
<tr>
<td>MPPfitcondloc=est_MIPPP_cond_loc(MPPcondloc$genMPP,MPPcondloc$r,hyper=1,L=L)</td>
</tr>
<tr>
<td>valid=floor(.1*L):L</td>
</tr>
<tr>
<td>gammaPME=apply(MPPfitcondloc$gen_gammas[valid,,2],mean)</td>
</tr>
<tr>
<td>GetStats(MPPfitcondloc$gen_gammas[valid,1])$CredibleSet</td>
</tr>
<tr>
<td>GetStats(MPPfitcondloc$gen_gammas[valid,2])$CredibleSet</td>
</tr>
</tbody>
</table>
GetStats(MPPfitcondloc$gen_gammas[valid,3])$CredibleSet

The posterior mean is given by $\gamma = [1.365227, 1.806892, 1.409466]$ with 95% credible sets $[0.9946535, 1.7567026]$, $[0.7288507, 3.0352708]$, and $[0.9401958, 1.8844168]$, respectively. Note that generic methods such as summary and plot can be applied to the MIPPP_fit object MPPfitcondloc returned from the est_MIPPP_cond_loc function call.

The results are summarized in figure 22. The left column presents the true surfaces for the MIPPP, whereas, the right column displays the corresponding estimates based on the fit obtained above.
Table 7: Comparison between the true values and the SEL posterior means for the component probabilities and means. The indices for the SEL fit corresponding to the truth are as follows: component 1 corresponds to component 3 in the truth, 2 to 2, 3 to 1, and 4 to 4. Note that all credible sets contain the true values (see the text).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>True Value</th>
<th>SEL posterior mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>0.2713132</td>
<td>0.1097771 (component 3 in truth)</td>
</tr>
<tr>
<td>$p_2$</td>
<td>0.3195929</td>
<td>0.394248 (component 2 in truth)</td>
</tr>
<tr>
<td>$p_3$</td>
<td>0.2116225</td>
<td>0.2954111 (component 1 in truth)</td>
</tr>
<tr>
<td>$p_4$</td>
<td>0.1974714</td>
<td>0.2005637 (component 4 in truth)</td>
</tr>
<tr>
<td>$\mu_1$</td>
<td>[-0.7737023, -0.3235898]</td>
<td>[1.575210,1.366443] (component 3 in truth)</td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>[-0.1849779,-0.9924813]</td>
<td>[-0.2587263,-1.0249956] (component 2 in truth)</td>
</tr>
<tr>
<td>$\mu_3$</td>
<td>[1.601975,1.405732]</td>
<td>[-0.09872487,-0.14601016] (component 1 in truth)</td>
</tr>
<tr>
<td>$\mu_4$</td>
<td>[1.673730,-1.427661]</td>
<td>[1.456489,-1.246632] (component 4 in truth)</td>
</tr>
</tbody>
</table>

Table 8: Model selection criteria for the simulated example. The AIC erroneously suggests 5 components, whereas, the BIC and ICLC correctly recommend a mixture with 4 components.

<table>
<thead>
<tr>
<th>$m$</th>
<th>AIC</th>
<th>BIC</th>
<th>ICLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3100.462</td>
<td>3125.785</td>
<td>3125.785</td>
</tr>
<tr>
<td>2</td>
<td>2866.115</td>
<td>2916.762</td>
<td>3145.518</td>
</tr>
<tr>
<td>3</td>
<td>2697.204</td>
<td>2773.175</td>
<td>3093.865</td>
</tr>
<tr>
<td>4</td>
<td>2655.917</td>
<td>2757.211</td>
<td>2869.450</td>
</tr>
<tr>
<td>5</td>
<td><strong>2644.390</strong></td>
<td>2771.007</td>
<td>2873.655</td>
</tr>
<tr>
<td>6</td>
<td>2655.630</td>
<td>2807.571</td>
<td>2965.292</td>
</tr>
</tbody>
</table>
Table 9: Frobenius norm for several methods of estimating the IPPP intensity surface. From the label switching free methods the average of the surfaces from the BDMCMC fit for MAP=4 components emerges as the best estimate. Regarding mixture deconvolution methods, the PME and KLE based on SEL permuted realizations from the DAMCMC fit emerge as the best surfaces.

<table>
<thead>
<tr>
<th>Estimation method</th>
<th>Frobenius norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>AoS (DAMCMC)</td>
<td><strong>604.5522</strong></td>
</tr>
<tr>
<td>MAPE (DAMCMC)</td>
<td>1020.9866</td>
</tr>
<tr>
<td>PME (SEL DAMCMC)</td>
<td><strong>771.1940</strong></td>
</tr>
<tr>
<td>KLE (SEL DAMCMC)</td>
<td><strong>776.1199</strong></td>
</tr>
<tr>
<td>AoS (BDMCMC)</td>
<td>603.0850</td>
</tr>
<tr>
<td>PME (BDMCMC)</td>
<td>1282.0500</td>
</tr>
<tr>
<td>MAPE (BDMCMC)</td>
<td>1034.0734</td>
</tr>
<tr>
<td>PME (IC BDMCMC)</td>
<td>914.9231</td>
</tr>
<tr>
<td>KLE (IC BDMCMC)</td>
<td>921.9302</td>
</tr>
<tr>
<td>PME (SEL BDMCMC)</td>
<td>802.9050</td>
</tr>
<tr>
<td>KLE (SEL BDMCMC)</td>
<td>810.8599</td>
</tr>
<tr>
<td>BMA (BDMCMC)</td>
<td><strong>624.5110</strong></td>
</tr>
</tbody>
</table>
Figure 1: Top row: 2d and 3d plots of the mixture density with 4 normal components. Bottom row: 2d and 3d plots of the corresponding mixture intensity surface with 4 normal components and $\lambda = 500$. All plots are over the window $\mathcal{W}_1 = [-2, 2] \times [-2, 2]$. 
Figure 2: True membership indicator variables. Component 1 has 180 events, 2 has 68 events, 3 has 154 events, and component 4 has 82.
Figure 3: 2d plots for the intensity of an IPPP over different windows.
Figure 4: 2d and 3d plots of the mixture intensity surface with 4 normal components and $\lambda = 500$, over the window $W = [-5, 5] \times [-5, 5]$. 
Figure 5: Visualizing a DAMCMC fit. Top row: 2d and 3d plots of the fitted intensity surface based on the posterior means. Middle row: posterior means of the membership indicator variables and trace plots for $p$. Bottom row: trace plots for the mixture component means.
Figure 6: Visualizing the DAMCMC fit after permuting the realizations using an identifiability constraint. Top row: 2d and 3d plots of the fitted intensity surface based on the posterior means. Middle row: posterior means of the membership indicator variables and trace plots for $p$. Bottom row: trace plots for the mixture component means.
Figure 7: Visualizing the DAMCMC fit after permuting the realizations using minimum Squared Error Loss. Top row: 2d and 3d plots of the fitted intensity surface based on the posterior means. Middle row: posterior means of the membership indicator variables and trace plots for $p$. Bottom row: trace plots for the mixture component means.
Figure 8: Comparison of the membership indicator variables; top-left: true allocation variables, top-right: posterior means (unpermuted labels), bottom-left: posterior means (IC permuted labels), and bottom-right: posterior means (SEL permuted labels). Note that the indices of components in the fits correspond to the indices in the truth as follows: component 1 corresponds to component 3 in the truth, 2 to 2, 3 to 1, and 4 to 4.
Figure 9: 2d plots of estimates of the IPPP intensity surface based on the DAMCMC fit. The left column presents the PME and the right column the KLE of the intensity surface, with each row corresponding to the unpermuted, IC permuted and SEL permuted realizations.
Figure 10: Top row: true IPPP intensity; Middle row: MAPE intensity; Bottom row: AoS intensity.
Figure 11: Assessing convergence for the third component probability $p_3$. Left column: running mean plots. Right column: autocorrelation plots. Note that since we have label switching using the unpermuted realizations the running mean plot does not indicate convergence (running mean is not almost a straight line), whereas, the IC and SEL permuted realizations indicate convergence.
Figure 12: Basic plots for a BDMCMC fit. Top row: distribution and trace plot for the number of components. Bottom row: 2d intensity and contour plots of the intensity surface of posterior means corresponding to realizations for the MAP number of components, which is 6. Clearly, we have label switching effects.
Figure 13: Basic plots for the BDMCMC fit after burn-in and the removal of the degenerate realizations. Top row: distribution of the number of components and 2d intensity plot of the intensity surface of posterior means based on the realizations for the MAP=4 number of components. Clearly, we have label switching effects.
Figure 14: Basic plots for the IC permuted BDMCMC fit after burn-in and the removal of the degenerate realizations. Top row: distribution of the number of components and 2d intensity plot of the intensity surface of posterior means based on the realizations for the MAP=4 number of components. Clearly, the IC permuted fit is better than the unpermuted fit.
Figure 15: Basic plots for the SEL permuted BDMCMC fit after burn-in and the removal of the degenerate realizations. Top row: distribution of the number of components and 2d intensity plot of the intensity surface of posterior means based on the realizations for the MAP=4 number of components. The SEL outperforms both the unpermuted and IC permuted fits.
Figure 16: Plots of the BDMCMC surface estimates in 2d. We present the true surface, the AoS and MAPE for MAP=4 number of components, the PME and KLE based on the SEL permuted realizations and the BMA.
Figure 17: Plots of the BDMCMC surface estimates in 3d. We present the true surface, the AoS and MAPE for MAP=4 number of components, the PME and KLE based on the SEL permuted realizations and the BMA.
Figure 18: Plots of the edge-corrected estimates of the IPPP intensity surface. We present the true surface, the AoS and the PME and KLE based on the SEL permuted realizations, the MAPE and the BMA surface in 2d. Note how all of the mass of a component is within the window.
Figure 19: Plots of the edge-corrected estimates of the IPPP intensity surface. We present the true surface, the AoS and the PME and KLE based on the SEL permuted realizations, the MAPE and the BMA surface in 3d. Note how all of the mass of a component is within the window.
Figure 20: Basic plots for a generated MIPPP conditioning on marks. First we display the events with circles denoting the different marks. The next 4 plots show the true ground processes along with the events corresponding to each mark value. The last plot illustrates the unmarked events as a general point pattern.
Figure 21: Conditioning on marks: displaying the 2d and 3d BMA surfaces of the ground processes for each mark, based on BDMCMC fits.
Figure 22: Conditioning on locations: the left column displays the true ground surface and probability fields for an MIPPP with 3 discrete marks. The right column displays the posterior estimates of the probability fields.