mpoints Documentation

Release alpha

Maxime Morariu-Patrichi, Mikko Pakkanen

Oct 10, 2018

Contents:

The *mpoints* package is a machine learning tool that implements the class of state-dependent Hawkes processes. Its key features include both simulation and estimation (statistical inference). It also contains a module with specialised plotting services.

State-dependent Hawkes processes belong to the class of hybrid marked point processes, a class that models the arrival in time of random events and their interaction with the state of a system.

We strongly recommend to first read the *tutorial*. It contains an introduction to this statistical model and illustrates the main services offered by the *mpoints* package.

For additional mathematical details, please consult the *documentation* and the *references*.

CHAPTER 1

Installation

The package can be easily installed via *pip*, a popular package management system for Python. In the terminal, simply enter the following command.

```
pip install mpoints
```

If you are using virtual environments (with conda), make sure that you install the package in the intended environment.

Note: when installing *mpoints* on Windows, you may be prompted to install Visual C++ Build Tools if this is not already installed.

CHAPTER 2

Tutorial

The `mpoints` package implements the class of state-dependent Hawkes processes. Its key features include both simulation and estimation (statistical inference). It also contains a module with specialised plot tools.

2.1 State-dependent Hawkes processes

State-dependent Hawkes processes model the arrival in time of random events and their interaction with the state of a system.

A state-dependent Hawkes process consists of an increasing sequence of random times (T_n), a sequence of random event types (E_n) in $\mathcal{E}$ (the set of possible event types) and a piecewise constant càdlàg state process $(X(t))$ in $\mathcal{X}$ (the set of possible states). Here, we assume that the event space $\mathcal{E}$ and state space $\mathcal{X}$ are finite, with respective sizes d_e and d_x.

2.1.1 Dynamics of events

Denote by $N_e(t)$ the number of events of type $e \in \mathcal{E}$ that have occured up to time t and by (T^e_n) the arrival times of events of type e. The arrival rate of events of type e, denoted by λ_e (its mathematical name is the intensity), is of the form

$$\lambda_e(t) = \nu_e + \sum_{e' \in \mathcal{E}} \int_{[0,t)} k_{e'e}(t-s, X(s)) dN_{e'}(s) = \nu_e + \sum_{e' \in \mathcal{E}} \sum_{n : T^{e'}_n < t} k_{e'e}(t - T^{e'}_n, X(T^{e'}_n)).$$

The non-negative kernel $k_{e'e}$ determines how events of type e' precipitate events of type e (by increasing their arrival rate). Notice that it depends on the state process. The kernels allow to introduce self- and cross-excitation effects. When all the kernels are null, the events behave like in a standard Poisson process with non-negative arrival rates $(\nu_e)_{e \in \mathcal{E}}$.

2.1.2 Dynamics of the state process

The state process can only jump at the event times (T_n). It jumps following transition probabilities that depend on both the current state and the event type. More precisely, denoting the history of $N := (N_e)_{e\in\mathcal{E}}$ and X just before time T_n by $\mathcal{F}^{N,X}_{T_n-}$, it statisfies

$$P(X(T_n) = x \mid E_n, \mathcal{F}^{N,X}_{T_n-}) = \phi_{E_n}(X(T_n-), x), \quad x \in \mathcal{X},$$

where $\phi := (\phi_e)_{e\in\mathcal{E}}$ is a family of transition probability matrices. In other words, $\phi_e(x', x)$ is the probability of transitioning from state x' to state x when an event of type e occurs.

Notice that the state process $(X(t))$ is fully determined by its values (X_n) at the event times (T_n), that is, $X_n := X(T_n)$.

Notice also that the counting process N and the state process X are fully coupled, in the sense that there is a two-way interaction: the self- and cross-excitation of N depend on X and the dynamics of X depend on N.

2.1.3 Exponential kernels

At the moment, the package implements only the case of exponential kernels, that is, we only consider parametric kernels of the form

$$k_{e'e}(t, x) = \alpha_{e'xe} \exp(-\beta_{e'xe} t), \quad t > 0, e', e \in \mathcal{E}, x \in \mathcal{X},$$

where the base rates ν and impact coefficients α are non-negative and the decay coefficients β are positive.

2.2 Setting up the model

From the `mpoints` package, we import the class `HybridHawkesExp` and the `plot_tools` module.

```
In [1]: import os
        import numpy as np
        from mpoints.hybrid_hawkes_exp import HybridHawkesExp
        from mpoints import plot_tools
        import seaborn  # for good-looking plots
        from IPython.display import set_matplotlib_formats  # set the figures format to svg
        set_matplotlib_formats('svg')
        %matplotlib inline
```

```
/Users/maximemorariu/anaconda/envs/py36/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56:
  from pandas.core import datetools
```

2.2.1 Set the meta-parameters

We set the dimension of the event space $\mathcal{E}$ and state space $\mathcal{X}$. We also name their elements.

```
In [2]: n_events = 2  # number of event types, $d_e$
        n_states = 2  # number of possible states, $d_x$
        events_labels = ['A', 'B']  # names of the event types
        states_labels = ['1', '2']  # names of the states
```

We initialise an instance of a state-dependent Hawkes process with exponential kernels.

```
In [3]: model = HybridHawkesExp(n_events, n_states, events_labels, states_labels)
```

2.2.2 Set the model parameters

We now need to input the parameters $\phi$, $\nu$, $\alpha$ and $\beta$.

```
In [4]: # The transition probabilities $\phi$
        phis = np.zeros((n_states, n_events, n_states))
        phis[0, 0, 0] = 0.7
        phis[0, 0, 1] = 0.3
        phis[1, 0, 0] = 0.6
        phis[1, 0, 1] = 0.4  # $\phi_0(1, 1)$, probability of transitioning from state 1 to 1 when a
        phis[0, 1, 0] = 0.2
        phis[0, 1, 1] = 0.8
        phis[1, 1, 0] = 0.4
        phis[1, 1, 1] = 0.6

        # The base rates $\nu$
        nus = np.ones(n_events)

        # The impact coefficients $\alpha$
        alphas = np.zeros((n_events, n_states, n_events))
        alphas[0, 0, 0] = 2
        alphas[0, 0, 1] = 1
        alphas[1, 0, 0] = 1
        alphas[1, 0, 1] = 4
        alphas[0, 1, 0] = 2
        alphas[0, 1, 1] = 10
        alphas[1, 1, 0] = 5
        alphas[1, 1, 1] = 1

        # The decay coefficients $\beta$
        betas = np.zeros((n_events, n_states, n_events))
        betas[0, 0, 0] = 4
        betas[0, 0, 1] = 12
        betas[1, 0, 0] = 14
        betas[1, 0, 1] = 10
        betas[0, 1, 0] = 14
        betas[0, 1, 1] = 15
        betas[1, 1, 0] = 10
        betas[1, 1, 1] = 14
```

Set the transition probabilities $\phi$.

```
In [5]: model.set_transition_probabilities(phis)
```

Set the parameters that govern the dynamics of the arrival rates, that is, $\nu$, $\alpha$ and $\beta$.

```
In [6]: model.set_hawkes_parameters(nus, alphas, betas)
```

2.2.3 Print the model parameters

The model paramaters can be conveniently converted to strings via the following methods.

```
In [7]: # Sequence of transition probability matrices, one per event type
        print(model.transition_matrix_to_string(phis))
```

```
[[ 0.7  0.3]
 [ 0.6  0.4]]
[[ 0.2  0.8]
 [ 0.4  0.6]]
```

```
In [8]: # Sequence of impact coefficient matrices, one per possible state
        print(model.impact_coefficients_to_string(alphas))
```

```
[[ 2.   1.]
 [ 1.   4.]]
[[  2.   10.]
 [  5.    1.]]
```

```
In [9]: # Sequence of decay coefficient matrices, one per possible state
        print(model.decay_coefficients_to_string(betas))
```

```
[[  4.   12.]
 [ 14.   10.]]
[[ 14.   15.]
 [ 10.   14.]]
```

2.3 Simulation

The model can now be simulated from time $t = 0$ to time $t = 7200$ (for two hours) as follows.

```
In [10]: time_start = 0
         time_end = 7200
         times, events, states = model.simulate(time_start, time_end)
```

Let's see how many events have occured in two hours.

```
In [11]: print('Number of events: ' + "{:,}".format(len(times)))
```

```
Number of events: 37,188
```

2.3.1 Plot a sample path

The sample path from time t_1 to time t_2 can be plotted as follows. In the upper subplot, we show the state process $(X(t))$. The dots indicate the arrival times (T_n) while their colour informs us on the event types (E_n). The lower subplot displays the arrival rates λ_e (intensities).

```
In [12]: t_1 = 0
         t_2 = 5
         seaborn.set()  # set the default seaborn style for figures
         fig, fig_array = plot_tools.sample_path(times, events, states, model, t_1, t_2)
```

2.3.2 Distribution of events and states

The joint distribution of (E_n, X_n) can be computed and plotted as follows.

```
In [13]: distribution = model.proportion_of_events_and_states(events, states, n_events, n_states)
         fig = plot_tools.discrete_distribution(distribution, v_labels = events_labels, h_labels = st
                                               figsize=(2, 2))
```

2.4 Statistical inference

Let's now imagine that `times`, `events` and `states` is some data that we want to analayse. We would like to fit a state-dependent Hawkes process to this sample path. Here, we should hopefully retrieve the original parameters `phis`, `nus`, `alphas` and `betas` that were used to generate this data.

Given this sample path, the model parameters can be estimated via maximum likelihood. It can be proven that the transition probabilities ϕ and the Hawkes parameters ν, α and β can be estimated independently, in spite of the strong coupling between N and X.

2.4.1 Estimate the transition probabilities

The transition probabilities ϕ are estimated as follows. Note that one can show that the maximum likelihood estimator is in fact given by the empirical transition probabilities.

```
In [14]: phis_hat = model.estimate_transition_probabilities(events, states)
```

The transition probabilities can be plotted as a heatmap. The estimated transition probabilities coincide indeed with the true ones.

```
In [15]: fig, fig_array = plot_tools.transition_probabilities(phis_hat, events_labels=events_labels,
                                                               figsize=(6, 3), left=0.12, right=0.88,
```

2.4.2 Estimate the Hawkes parameters

The package comes with a built-in method that searches for the parameters ν, α and β that maximise the likelihood of events. It is based on the `scipy.optimize.minimize` module and, by default, employs a conjugate gradient method (other methods from `scipy.optimize.minimize` can be called, see documentation). More advanced users can apply their own optimisation algorithm by calling directly the methods that compute the log-likelihood or partial log-likelihoods and the gradient or partial gradients (again, see the documentation).

```
In [16]: opt_result, initial_guess, initial_guess_kind = model.estimate_hawkes_parameters(times, ever
                                                                                         time_start,
```

The method returns a `scipy.optimize.OptimizeResult` instance which contains the maximum likelihood estimate as a 1D numpy array. One can go from this 1D array to the usual parameter format as follows.

```
In [17]: # the maximum likelihood estimate in a 1D array
         mle_estimate = opt_result.x
         # tranform it to the usual format
         nus_hat, alphas_hat, betas_hat = model.array_to_parameters(mle_estimate, n_events, n_states)
```

We check that our estimate of the base rates ν is close to the original value `nus`.

```
In [18]: print(nus)
         print(nus_hat)
```

```
[ 1.  1.]
[ 0.98175138  0.98502245]
```

The kernels can be visualised by plotting the cumulative excitation functions

$$t \mapsto ||k_{e'e}(\cdot, x)||_{1,t} := \int_0^t k_{e'e}(s, x)ds, \quad e', e \in \mathcal{E}, x \in \mathcal{X},$$

which provide a convenient visualisation of the magnitude of the self- and cross-excitation effects and the effective timescales at which they occur. One can interpret $||k_{e'e}(\cdot, x)||_{1,t}$ as the average number of events of type e that are directly triggered by an event of type e' within t seconds of its occurrence, under state x. We plot these functions for the estimated and true kernels.

```
In [19]: # Estimated kernels in full line
         fig, fig_array = plot_tools.kernels_exp(alphas_hat, betas_hat, events_labels=events_labels,
                                                 log_timescale=True, figsize=(7, 4))
         # True kernels in dashed line
         fig, fig_array = plot_tools.kernels_exp(alphas, betas,
                                                 tmax=1, log_timescale=True, fig=fig, fig_array=fig_a
                                                 left=0.12, right=0.88, bottom=0.15, top=0.85, ymax=(
         # Add x and y labels
         txt = fig.text(0.5, 0.02, 'Time (seconds)', ha='center', fontsize=16)
         txt = fig.text(0.01, 0.5, r'$||\hat{k}||_{1, t}$', va='center', rotation='vertical', fontsiz
```

For example, the top right subplot shows the kernels $k_{AB}(\cdot, x)$ for the two different states $x = 1$ and $x = 2$. We were indeed able to approximately retrieve the true paramaters from the sample path, thanks to the maximum likelihood princinple.

2.4.3 Goodness-of-fit

Define the event residuals r^e_n by

$$r^e_n := \int_{t^e_{n-1}}^{t^e_n} \lambda_e(t)dt, \quad e \in \mathcal{E},$$

where t^e_n is the time when the n th event of type e occurred (that is, t^e_n is the realisation of random variable T^e_n).

Let's compute the sequences of residuals (one per event type) under the estimated model. We first update the model paramaeters with the estimated ones.

```
In [20]: model.set_transition_probabilities(phis_hat)
         model.set_hawkes_parameters(nus_hat, alphas_hat, betas_hat)
```

The event residuals can then be computed as follows.

```
In [21]: residuals = model.compute_events_residuals(times, events, states, time_start)
```

Under the assumption that the sample path was generated by this estimated model, the event residuals $(r^e_n)_{n\in\mathbb{N},e\in\mathcal{E}}$ are the realisation of i.i.d standard unit rate exponential random variables. Hence, the goodness-of-fit of our estimated model can be assessed by analysing the distribution of the event residuals.

For comparison, we also compute the event residuals under a naive Poisson model.

```
In [22]: n_states_poisson = 1  # no state process under this naive model
         model_poisson = HybridHawkesExp(n_events, n_states_poisson, events_labels, ['no state'])
         # the transition probabilities are then trivial
         model_poisson.set_transition_probabilities(np.ones((n_states_poisson, n_events, n_states_poi
         # compute the mean arrival rate and use it as the base rate
         nus_poisson = np.zeros(n_events)
         for e in range(n_events):
             nus_poisson[e] = (events == e).sum() / (time_end - time_start)
         # impact coefficients set to null to get a Poisson process
         alphas_poisson = np.zeros((n_events, n_states_poisson, n_events))
         # deceay coefficents, their value does not matter here
         betas_poisson = np.ones((n_events, n_states_poisson, n_events))
         # set the parameters that govern the arrival rates of events (intensities)
         model_poisson.set_hawkes_parameters(nus_poisson, alphas_poisson, betas_poisson)
         # compute the event residuals under this naive model
```

```
        states_poisson = np.zeros(len(times), dtype='int')  # we create a fake trivial state proces
        residuals_poisson = model_poisson.compute_events_residuals(times, events, states_poisson, t
```

For every event type $e \in \mathcal{E}$, we compare the distribution of (r_n^e) to the standard exponential distirbution via a qq-plot. We do this under both the state-dependent-Hawkes-proces model and the Poisson model.

```
In [23]: model_labels = ['Hawkes', 'Poisson']
         fig, fig_array = plot_tools.qq_plot([residuals, residuals_poisson], shape=(1, n_events), lab
                                             model_labels=model_labels,
                                             figsize=(8, 4), left=0.085, right=0.915, bottom=0.15, to
```

We also plot the correlogram of the event residuals time series $(r_n) := (r_n^1, \dots, r_n^{d_e})$ as a test of the mutual independence.

```
In [24]: fig, fig_array = plot_tools.correlogram([residuals, residuals_poisson], labels=events_label
                                                 model_labels=model_labels,
                                                 figsize=(6, 5), left=0.12, right=0.88, bottom=0.12,
```

As expected, the state-dependent Hawkes process provides an (almost) perfect fit but the Poisson model does not.

A similar study can be performed with the total residuals r_n^{ex}, defined by

$$r_n^{ex} := \int_{t_{n-1}^{ex}}^{t_n^{ex}} \phi_e(X(t), x)\lambda_e(t)dt,$$

where t_n^{ex} is the time when the n th event of type e after which the state is x occurred.

```
In [25]: residuals_total = model.compute_total_residuals(times, events, states, time_start)
         product_labels = model.generate_product_labels()
         fig, fig_array = plot_tools.qq_plot(residuals_total, shape=(n_events, n_states), labels=proc
                                             figsize=(8, 4), left=0.085, right=0.915, bottom=0.15, to
```

```
In [26]: fig, fig_array = plot_tools.correlogram(residuals_total, labels=product_labels, n_lags=10,
                                                left=0.1, right=0.9, bottom=0.12, top=0.88)
```

2.5 Application to high-frequency financial data

State-dependent Hawkes processes were applied to high-frequency financial data in the following paper.

Morariu-Patrichi, M. and Pakkanen, M. S. (2018). State-dependent Hawkes processes and their application to limit order book modelling. Preprint, available at https://arxiv.org/abs/1809.08060.

CHAPTER 3

C extension

The crucial algorithms of this package (simulation, likelihood and gradient computations, computation of residuals) are in fact implemented in C via Cython. The class `HybridHawkesExp` wraps a C extension that is compiled when the package is installed with *pip*.

Without the increase in computational speed provided by the C extension, the statistical inference of state-dependent Hawkes processes would be impractical.

The Cython code that generated the C extension is available in the GitHub repository (the .pyx file).

CHAPTER 4

Documentation

4.1 hybrid_hawkes_exp module

class mpoints.hybrid_hawkes_exp.**HybridHawkesExp**(*number_of_event_types*, *number_of_states*, *events_labels*, *states_labels*)

Bases: object

This class implements state-dependent Hawkes processes with exponential kernels, a subclass of hybrid marked point processes. The main features it provides include simulation and statistical inference (estimation).

Parameters

- **number_of_event_types** (*int*) – number of different event types.
- **number_of_states** (*int*) – number of possible states.
- **events_labels** (*list of strings*) – names of the different event types.
- **states_labels** (*list of strings*) – names of the possible states.

static array_to_parameters(*array*, *number_of_event_types_1*, *number_of_states*, *number_of_event_types_2=0*)

Retrieves the parameters (ν, α, β) from a 1D array. It is NOT assumed that the length of the 1st and 3rd dimensions of the arrays $(\alpha_{e'xe})$ and $(\beta_{e'xe})$ are equal. For instance, in log_likelihood_of_events_partial(), only of subgroup of the parameters (ν, α, β) are required.

Parameters

- **array** (*1D numpy array*) – an array containing the parameters (ν, α, β).
- **number_of_event_types_1** (*int*) – length of the first dimension of $(\alpha_{e'xe})$ and $(\beta_{e'xe})$.
- **number_of_states** (*int*) – number of possible states in the state-dependent Hawkes process, length of the second dimension.

- **number_of_event_types_2** (*int*) – length of the third dimension of $(\alpha_{e'xe})$ and $(\beta_{e'xe})$. It is also the length of ν. If set to zero, we assume that *array* contains ALL the parameters and not only a subgroup, meaning that 1st and 3rd dimensions of the arrays $(\alpha_{e'xe})$ and $(\beta_{e'xe})$ are equal.

Return type 1D numpy array, 3D numpy array, 3D numpy array

Returns the parameters (ν, α, β).

compute_events_residuals(*times*, *events*, *states*, *time_start*, *initial_partial_sums=0*)

Computes the events residuals r_n^e defined by

$$r_n^e := \int_{t_{n-1}^e}^{t_n^e} \lambda_e(t)dt,$$

where t_n^e is the time when the *n* th event of type *e* occurred. The methods wraps a C implementation that was obtained via Cython.

Parameters

- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **time_start** (*float*) – the time at which we consider that the process started, prior times are treated as an initial condition.
- **initial_partial_sums** (*3D numpy array*) – the initial condition can also be given implicitly via the partial sums $S_{e',x,e}(-\infty, \text{time}_start]$.

Return type list of 1D numpy arrays

Returns the *e* th element of the list is the sequence (r_n^e) corresponding to the event type *e*.

compute_partial_sums(*times*, *events*, *states*, *time_end*, *initial_partial_sums=None*, *time_initial_condition=None*)

Computes the partial sums

$$S_{e'xe}(-\infty, T] := \alpha_{e'xe} \sum_{n:t_n^{e'x} \leq T} \exp(-\beta_{e'xe}(T - t_n^{e'x})),$$

where *e* and *e'* are event types, *x* is a possible state, $t_n^{e'x}$ is the time when the *n* th event of type *e'* after which the state is *x* occurred. The collection of partial sums $(S_{e'xe})$ at time T encodes the history of the state-dependent Hawkes process up to time T. It can for example be used to simulate the process from time T. In `simulate()`, one can pass the partial sums as an initial condition instead of the entire history.

Parameters

- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **time_end** (*float*) – T, the time up to which the partial sums are computed.
- **initial_partial_sums** (*3D numpy array of float*) – an initial condition (events that occur before *times*) can be given implicitly as partial sums.

- **time_initial_condition** (*float*) – the time up to which the partial sums given as an initial condition were computed.

Return type 3D numpy array of float

Returns the partial sums, *array[e', x, e]* corresponds to $S_{e'x'e}$.

compute_total_residuals(*times*, *events*, *states*, *time_start*, *initial_partial_sums=0*, *initial_state=0*)

Computes the total residuals r^{ex}_n defined by

$$r^{ex}_n := \int_{t^{ex}_{n-1}}^{t^{ex}_n} \phi_e(X(t), x)\lambda_e(t)dt,$$

where t^{ex}_n is the time when the *n* th event of type *e* after which the state is *x* occurred. The methods wraps a C implementation that was obtained via Cython.

Parameters

- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **time_start** (*float*) – the time at which we consider that the process started, prior times are treated as an initial condition.
- **initial_partial_sums** (*3D numpy array*) – the initial condition can also be given implicitly via the partial sums $S_{e',x,e}(-\infty, \text{time}_s tart]$. **initial_state** (*int*) – if there are no event times before *time_start*, this is used as th

Return type list of 1D numpy arrays

Returns the sequence (r^{ex}_n) is the *x + e * number_of_states* th element in the list.

static decay_coefficients_to_string(*decay_coefficients*)

Transforms the decay coefficients β to a string that can be printed in the console or a text file.

Parameters **decay_coefficients** (*3D numpy array*) – the decay coefficients β.

Return type string

Returns sequence of matrices $(\beta_{\cdot x \cdot})$, one per possible state *x*.

estimate_hawkes_parameters(*times*, *events*, *states*, *time_start*, *time_end*, *maximum_number_of_iterations=2000*, *method='TNC'*, *parameters_lower_bound=1e-06*, *parameters_upper_bound=None*, *given_guesses=[]*, *number_of_random_guesses=1*, *min_decay_coefficient=0.5*, *max_decay_coefficient=100*, *parallel_estimation=True*)

Estimates the parameters of the intensities (arrival rates) of events, i.e., (ν, α, β). Estimation if performed via maximum likelihood. This method uses the *scipy.minimize* library.

Parameters

- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.

- **`time_start`** (*`float`*) – the time at which we consider that the process started, prior times are treated as an initial condition.
- **`time_end`** (*`float`*) – the time at which we stopped to record the process.
- **`maximum_number_of_iterations`** (*`int`*) – will be passed to the *maxiter* argument in *scipy.minimize*. Depending on *method*, it is the maximum number of iterations or function evaluations.
- **`method`** (*`string`*) – the optimisation method used in *scipy.minimize*.
- **`parameters_lower_bound`** (*`float`*) – lower bound on all the parameters.
- **`parameters_upper_bound`** (*`float`*) – upper bound on all the parameters.
- **`given_guesses`** (*`list of 1D numpy array`*) – every member *x* is an initial guess on the parameters. For every *x*, we attempt to maximise the likelihood starting from *x*. One can go from *x* to (ν, α, β) and vice versa using `array_to_parameters()` and `parameters_to_array()`. We retain the solution that gives the highest likelihood.
- **`number_of_random_guesses`** (*`int`*) – the method can also generate random initial guesses.
- **`min_decay_coefficient`** (*`numpy array or float`*) – defines how a random guess is generated.
- **`max_decay_coefficient`** (*`numpy array of float`*) – a random guess on $\beta_{e'xe}$ is generated uniformly in the interval [*min_decay_coefficient[e',x,e]*, *max_decay_coefficient[e',x.e]*] but on a logarithmic scale.
- **`parallel_estimation`** (*`boolean`*) – the MLE problem can be decomposed into d_e independent optimisation problems, where d_e is the number of event types. When True, each problem is solved independently. In this case, the limit on the number of iterations or function evaluations is applied independently to each sub-problem.

Return type scipy.optimize.OptimizerResult, 1D numpy array, string

Returns The first object is the optimisation result and contains the maximum likelihood estimate along with additional information on the optimisation routine. The second object contains the initial guess that resulted in the highest likelihood after running the optimisation procedure. The third object indicates the nature of this initial guess ('random' or 'given').

`estimate_transition_probabilities`(*events*, *states*)

Estimates the transition probabilities ϕ of the state process from the data. This method returns the maximum likelihood estimate. One can prove that it coincides with the empirical transition probabilities.

Parameters

- **`events`** (*`1D array of int`*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **`states`** (*`1D array of int`*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.

Return type 3D array

Returns the estimated transition probabilities ϕ.

`generate_base_rates_labels`()

Produces labels for the base rates ν. This uses the events labels of the model.

Return type list of string

Returns *list[e]* returns a label for ν_e.

generate_decay_coefficients_labels()

Produces labels for the decay coefficients β. This uses the events and states labels of the model.

Return type list of list of list of string

Returns *list[e'][x][e]* returns a label for $\beta_{e'xe}$.

generate_impact_coefficients_labels()

Produces labels for the impact coefficients α. This uses the events and states labels of the model.

Return type list of list of list of string

Returns *list[e'][x][e]* returns a label for $\alpha_{e'xe}$.

generate_product_labels()

Produces labels for all the couples of events types and possible states. This uses the events and states labels of the model.

Return type list of strings

Returns the label for the couple *(e, x)* is given by by *list[n]* where $n = x + e \times \text{number}_o f_e vent_t ypes$.

gradient(*parameters*, *times*, *events*, *states*, *time_start*, *time_end*)

Computes the gradient of the log-likelihood l with respect to the parameters (ν, α, β). The method wraps a C implementation that was obtained via Cython.

Parameters

- **parameters** (*1D numpy array*) – the parameters (ν, α, β) put into a single array. Go from *parameters* to (ν, α, β) and vice versa using `array_to_parameters()` and `parameters_to_array()`.
- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **time_start** (*float*) – t_0, the time at which we consider that the process started, prior times are treated as an initial condition.
- **time_end** (*float*) – T, the time at which we stopped to record the process.

Return type float

Returns the gradient of the log-likelihood l.

gradient_partial(*event_type*, *parameters*, *times*, *events*, *states*, *time_start*, *time_end*)

Computes the gradient of the partial log-likelihood l_e with respect to the parameters (ν, α, β) that govern λ_e, the intensity of events of type *e*. The method wraps a C implementation that was obtained via Cython.

Parameters

- **event_type** (*int*) – *e*, the event type for which we want to compute the gradient of the partial log-likelihood l_e.
- **parameters** (*1D numpy array*) – only the parameters (ν, α, β) that govern λ_e put into a single array. Go from *parameters* to (ν, α, β) and vice versa using `array_to_parameters()` and `parameters_to_array()`.
- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.

- **`states`** (*`1D numpy array of int`*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **`time_start`** (*`float`*) – t_0, the time at which we consider that the process started, prior times are treated as an initial condition.
- **`time_end`** (*`float`*) – T, the time at which we stopped to record the process.

Return type float

Returns the gradient of the partial log-likelihood l_e.

static `impact_coefficients_to_string`(*impact_coefficients*)
Transforms the impact coefficients α to a string that can be printed in the console or a text file.

Parameters **`impact_coefficients`** (*`3D numpy array`*) – the impact coefficients α.

Return type string

Returns sequence of matrices $(\alpha_{\cdot x \cdot})$, one per possible state *x*.

`intensities_of_events`(*partial_sums*)
Computes the intensities, given the partial sums $S_{e'xe}$.

Parameters **`partial_sums`** (*`3D numpy array of float`*) – the partial sums $S_{e',x',e}$ at the considered time.

Return type 1D numpy array of float

Returns the intensities, *array[e]* is the value of λ_e at the considered time.

`intensities_of_events_at_times`(*compute_times*, *times*, *events*, *states*)
Computes the intensities at the *compute_times* given a realisation of the state-dependent Hawkes process.

Parameters

- **`compute_times`** (*`1D numpy array of float`*) – the times at which the intensities will be computed.
- **`times`** (*`1D numpy array of float`*) – the times at which events occur.
- **`events`** (*`1D numpy array of int`*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **`states`** (*`1D numpy array of int`*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.

Return type 1D numpy array of float, 2D numpy array of float

Returns the first array contains both the *compute_times* and the event times of the given realisation, in increasing order. Note that the event times are put there twice: the intensities are computed just before and right after the event times. The second array gives the intensities at the times of the first array. *array2[e,n]* is the intensity of events of type *e* at time *array1[n]*.

`intensity_of_event`(*event_type*, *partial_sums*)
Computes the intensity of events of type *event_type*, given the partial sums $S_{e'xe}$.

Parameters

- **`event_type`** (*`int`*) – the event type for which the intensity is computed.
- **`partial_sums`** (*`3D numpy array of float`*) – the partial sums $S_{e',x',e}$ at the considered time.

Return type float

Returns the value of λ_e at the considered time.

static kernel_at_time(*time*, *alpha*, *beta*)
Evaluates the kernel of the model at the given time with the given parameters.

Parameters

- **time** (*float*) – the positive time t.
- **alpha** (*float*) – a non-negative α.
- **beta** (*float*) – a positive β.

Return type float

Returns $\alpha \exp(-\beta t)$.

log_likelihood_of_events(*parameters*, *times*, *events*, *states*, *time_start*, *time_end*)
Computes the log-likelihood of the observed times and event types under the assumption that they are the realisation of a state-dependent Hawkes process with the given parameters. The log-likelihood is given by

$$l := \sum_{n : t_0 < t_n \leq T} \lambda_{e_n}(t_n) - \sum_e \int_{t_0}^{T} \lambda_e(t) dt,$$

where (t_n) and (e_n) are the sequences of event times and event types, respectively. The method wraps a C implementation that was obtained via Cython.

Parameters

- **parameters** (*1D numpy array*) – the parameters (ν, α, β) put into a single array. Go from *parameters* to (ν, α, β) and vice versa using `array_to_parameters()` and `parameters_to_array()`.
- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **time_start** (*float*) – t_0, the time at which we consider that the process started, prior times are treated as an initial condition.
- **time_end** (*float*) – T, the time at which we stopped to record the process.

Return type float

Returns the log-likelihood l.

log_likelihood_of_events_partial(*event_type*, *parameters*, *times*, *events*, *states*, *time_start*, *time_end*)
Computes the log-likelihood of the arrival times of events of the given type under the assumption that they are the realisation of a state-dependent Hawkes process with the given parameters. These parameters include only those that govern λ_e, the intensity of events of type *e*. For example, among the base rates ν, only ν_e should be contained in *parameters*. This partial log-likelihood is given by

$$l_e := \sum_{n : t_0 < t_n^e \leq T} \lambda_e(t_n^e) - \int_{t_0}^{T} \lambda_e(t) dt,$$

where (t_n^e) are the arrival times of events of the given type *e*. The method wraps a C implementation that was obtained via Cython.

Parameters

- **event_type** (*int*) – *e*, the event type for which we want to compute the partial log-likelihood l_e.
- **parameters** (*1D numpy array*) – only the parameters (ν, α, β) that govern λ_e put into a single array. Go from *parameters* to (ν, α, β) and vice versa using `array_to_parameters()` and `parameters_to_array()`.
- **times** (*1D numpy array of float*) – the times at which events occur.
- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **time_start** (*float*) – t_0, the time at which we consider that the process started, prior times are treated as an initial condition.
- **time_end** (*float*) – T, the time at which we stopped to record the process.

Return type float

Returns the partial log-likelihood l_e.

static parameters_to_array (*base_rates*, *impact_coefficients*, *decay_coefficients*)

Puts the model parameters (ν, α, β) into a one dimensional array.

Parameters

- **base_rates** (*1D numpy array*) – the collection (ν_e).
- **impact_coefficients** (*3D numpy array*) – the collection $(\alpha_{e'xe})$.
- **decay_coefficients** (*3D numpy array*) – the collection $(\beta_{e'xe})$.

Return type 1D numpy array

Returns the parameters put into a single 1D array.

static proportion_of_events_and_states (*events*, *states*, *number_of_event_types*, *number_of_states*)

Computes the empirical distribution of the events and states.

Parameters

- **events** (*1D numpy array of int*) – the sequence of event types, *events[n]* is the event type of the *n* th event.
- **states** (*1D numpy array of int*) – the sequence of states, *states[n]* is the new state of the system following the *n* th event.
- **number_of_event_types** (*int*) – number of different event types.
- **number_of_states** (*int*) – number of possible states.

Return type 2D numpy array of float

Returns *array[e,x]* is the percentage of events of type *e* after which the state is *x*.

set_hawkes_parameters (*base_rates*, *impact_coefficients*, *decay_coefficients*)

Fixes the parameters (ν, α, β) that define the intensities (arrival rates) of events. The are used in `simulate()`, `compute_events_residuals()` and `compute_total_residuals()`.

Parameters

- **base_rates** (*1D numpy array*) – one base rate ν_e per event type e.
- **impact_coefficients** (*3D numpy array*) – the alphas $\alpha_{e'xe}$.

- **decay_coefficients** (*3D numpy array*) – the betas $\beta_{e'xe}$.

Returns

set_transition_probabilities(*transition_probabilities*)

Fixes the transition probabilities ϕ of the state-dependent Hawkes process. The are used to `simulate()` and `compute_total_residuals()`.

Parameters **transition_probabilities** (*3D numpy array*) – shape should be (d_x, d_e, d_x) where d_e and d_x are the number of event types and states, respectively. The entry i, j, k is the probability of going from state i to state k when an event of type j occurs.

Returns

simulate(*time_start*, *time_end*, *initial_condition_times=[]*, *initial_condition_events=[]*, *initial_condition_states=[]*, *initial_partial_sums=0*, *initial_state=0*, *max_number_of_events=1000000*)

Simulates a sample path of the state-dependent Hawkes process. The methods wraps a C implementation that was obtained via Cython.

Parameters

- **time_start** (*float*) – time at which the simulation starts.
- **time_end** (*float*) – time at which the simulation ends.
- **initial_condition_times** (*array*) – times of events before and including *time_start*.
- **initial_condition_events** (*array of int*) – types of the events that occurred at *initial_condition_times*.
- **initial_condition_states** (*array of int*) – values of the state process just after the *initial_condition_times*.
- **initial_partial_sums** (*3D numpy array*) – the initial condition can also be given implicitly via the partial sums $S_{e',x,e}(-\infty, \text{time}_start]$. **initial_state** (*int*) – if there are no event times before *time_start*, this is used as th

Return type array, array of int, array of int

Returns the times at which the events occur, their types and the values of the state process right after each event. Note that these include the initial condition as well.

static transition_matrix_to_string(*transition_probabilities*)

Transforms the transition probabilities ϕ to a string that can be printed in the console or a text file.

Parameters **transition_probabilities** (*3D numpy array*) – the transition probabilities ϕ.

Return type string

Returns sequence of transition probability matrices (ϕ_e), one per event type *e*.

4.2 plot_tools module

mpoints.plot_tools.**correlogram**(*residuals*, *path=''*, *fig_name='correlogram.pdf'*, *title=None*, *labels=None*, *model_labels=None*, *palette=None*, *n_lags=50*, *figsize=(8, 6)*, *size_labels=16*, *size_ticks=14*, *size_legend=16*, *bottom=None*, *top=None*, *left=None*, *right=None*, *savefig=False*)

Correlogram of residuals.

Parameters

- **residuals** (*list*) – list of lists (one list of residuals per event type) or list of lists of lists when multiple models are compared (one list of lists per model).
- **path** (*string*) – where the figure is saved.
- **fig_name** (*string*) – name of the file.
- **title** (*string*) – suptitle.
- **labels** (*list of strings*) – labels of the event types.
- **model_labels** (*list of strings*) – names of the different considered models.
- **palette** (*list of colours*) – color palette, one color per model.
- **n_lags** (*int*) – number of lags to plot.
- **figsize** (*(int, int)*) – tuple (width, height).
- **size_labels** (*int*) – fontsize of labels.
- **size_ticks** (*int*) – fontsize of tick labels.
- **legend_size** (*int*) – fontsize of the legend.
- **bottom** (*float*) – between 0 and 1, adjusts the bottom margin, see matplotlib subplots_adjust.
- **top** (*float*) – between 0 and 1, adjusts the top margin, see matplotlib subplots_adjust.
- **left** (*float*) – between 0 and 1, adjusts the left margin, see matplotlib subplots_adjust.
- **right** (*float*) – between 0 and 1, adjusts the right margin, see matplotlib subplots_adjust.
- **savefig** (*boolean*) – set to True to save the figure.

Return type Figure, array of Axes

Returns the figure and array of figures (see matplotlib).

mpoints.plot_tools.**discrete_distribution**(*probabilities*, *path=''*, *fig_name='distribution_events_states.pdf'*, *v_labels=None*, *h_labels=None*, *title=None*, *color_map=None*, *figsize=(12, 6)*, *size_labels=16*, *size_values=14*, *bottom=None*, *top=None*, *left=None*, *right=None*, *savefig=False*, *usetex=False*)

Annotated heatmap of a given discrete distribution with 2 dimensions.

Parameters

- **probabilities** (*2D array*) – the 2D discrete distribution.
- **path** (*string*) – where the figure is saved.
- **fig_name** (*string*) – name of the file.
- **v_labels** (*list of strings*) – labels for the first dimension (vertical).
- **h_labels** (*list of strings*) – labels for the second dimension (horizontal).
- **title** (*string*) – suptitle.
- **color_map** – color map for the heatmap, see seaborn documentation.
- **figsize** (*(int, int)*) – tuple (width, height).

- **size_labels** (*int*) – fontsize of labels.
- **size_values** (*int*) – fontsize of the annotations on top of the heatmap.
- **bottom** (*float*) – between 0 and 1, adjusts the bottom margin, see matplotlib subplots_adjust.
- **top** (*float*) – between 0 and 1, adjusts the top margin, see matplotlib subplots_adjust.
- **left** (*float*) – between 0 and 1, adjusts the left margin, see matplotlib subplots_adjust.
- **right** (*float*) – between 0 and 1, adjusts the right margin, see matplotlib subplots_adjust.
- **savefig** (*boolean*) – set to True to save the figure.
- **usetex** (*boolean*) – set to True if matplolib figure is rendered with TeX.

Return type Figure

Returns the figure (see matplotlib).

`mpoints.plot_tools.`**`kernels_exp`**(*impact_coefficients*, *decay_coefficients*, *events_labels=None*, *states_labels=None*, *path=”*, *fig_name='kernels.pdf'*, *title=None*, *palette=None*, *figsize=(9, 7)*, *size_labels=16*, *size_values=14*, *size_legend=16*, *bottom=None*, *top=None*, *left=None*, *right=None*, *savefig=False*, *fig_array=None*, *fig=None*, *tmin=None*, *tmax=None*, *npoints=500*, *ymax=None*, *alpha=1*, *legend_pos=0*, *log_timescale=True*, *ls='-'*)

Plots the kernels of a state-dependent Hawkes process. Here the kernels are assumed to be exponential, that is, $k_{e'e}(t,x) = \alpha_{e'xe}\exp(-\beta_{e'xe}t)$. We plot the functions

$$t \mapsto ||k_{e'e}(\cdot,x)||_{1,t} := \int_0^t k_{e'e}(s,x)ds.$$

The quantity $||k_{e'e}(\cdot,x)||_{1,t}$ can be interpreted as the average number of events of type e that are directly precipitated by an event of type e' within t units of time, under state x. There is a subplot for each couple of event types (e',e). In each subplot, there is a curve for each possible state x.

Parameters

- **impact_coefficients** (*3D array*) – the alphas $\alpha_{e'xe}$.
- **decay_coefficients** (*3D array*) – the betas $\beta_{e'xe}$.
- **events_labels** (*list of strings*) – labels of the event types.
- **states_labels** (*list of strings*) – labels of the states.
- **path** (*string*) – where the figure is saved.
- **fig_name** (*string*) – name of the file.
- **title** (*string*) – suptitle.
- **palette** (*list of colours*) – color palette, one color per state x.
- **figsize** (*(int, int)*) – tuple (width, height).
- **size_labels** (*int*) – fontsize of labels.
- **size_values** (*int*) – fontsize of tick labels.
- **size_legend** (*int*) – fontsize of the legend.
- **bottom** (*float*) – between 0 and 1, adjusts the bottom margin, see matplotlib subplots_adjust.

- **top** (*float*) – between 0 and 1, adjusts the top margin, see matplotlib subplots_adjust.
- **left** (*float*) – between 0 and 1, adjusts the left margin, see matplotlib subplots_adjust.
- **right** (*float*) – between 0 and 1, adjusts the right margin, see matplotlib subplots_adjust.
- **savefig** (*boolean*) – set to True to save the figure.
- **fig_array** (*array of Axes*) – fig_array, where to plot the kernels (see matplotlib).
- **fig** (*Figure*) – figure, where to plot the figure (see matplotlib).
- **tmin** (*float*) – we plot over the time interval [*tmin*, *tmax*].
- **tmax** (*float*) – we plot over the time interval [*tmin*, *tmax*].
- **npoints** (*int*) – number of points used to plot.
- **ymax** (*float*) – upper limit of the y axis.
- **alpha** (*float*) – between 0 and 1, transparency of the curves.
- **legend_pos** (*int*) – position of the legend in the array of figures.
- **log_timescale** (*boolean*) – set to False to plot with a linear timescale.g
- **ls** (*string*) – the linestyle (see matplotlib).

Return type Figure, array of Axes

Returns the figure and array of figures (see matplotlib).

mpoints.plot_tools.**qq_plot**(*residuals*, *shape=None*, *path=”*, *fig_name='qq_plot.pdf'*, *log=False*, *q_min=0.01*, *q_max=0.99*, *number_of_quantiles=100*, *title=None*, *labels=None*, *model_labels=None*, *palette=None*, *figsize=(12, 6)*, *size_labels=16*, *size_ticks=14*, *legend_size=16*, *bottom=0.12*, *top=0.93*, *left=0.08*, *right=0.92*, *savefig=False*, *leg_pos=0*)

Qq-plot of residuals.

Parameters

- **residuals** (*list*) – list of lists (one list of residuals per event type) or list of lists of lists when multiple models are compared (one list of lists per model).
- **shape** (*(int, int)*) – 2D-tuple (number of rows, number of columns), shape of the array of figures.
- **path** (*string*) – where the figure is saved.
- **fig_name** (*string*) – name of the file.
- **log** (*boolean*) – set to True for qq-plots with log-scale.
- **q_min** (*float*) – smallest quantile to plot (e.g., 0.01 for 1%).
- **q_max** (*float*) – largest quantile to plot.
- **number_of_quantiles** (*int*) – number of points used to plot.
- **title** (*string*) – suptitle.
- **labels** (*list of strings*) – labels of the event types.
- **model_labels** (*list of strings*) – names of the different considered models.
- **palette** (*list of colours*) – color palette, one color per model.
- **figsize** (*(int, int)*) – tuple (width, height).
- **size_labels** (*int*) – fontsize of labels.

- **`size_ticks`** (*`int`*) – fontsize of tick labels.
- **`legend_size`** (*`int`*) – fontsize of the legend.
- **`bottom`** (*`float`*) – between 0 and 1, adjusts the bottom margin, see matplotlib subplots_adjust.
- **`top`** (*`float`*) – between 0 and 1, adjusts the top margin, see matplotlib subplots_adjust.
- **`left`** (*`float`*) – between 0 and 1, adjusts the left margin, see matplotlib subplots_adjust.
- **`right`** (*`float`*) – between 0 and 1, adjusts the right margin, see matplotlib subplots_adjust.
- **`savefig`** (*`boolean`*) – set to True to save the figure.
- **`leg_pos`** (*`int`*) – position of the legend in the array of figures.

Return type Figure, array of Axes

Returns the figure and array of figures (see matplotlib).

`mpoints.plot_tools.`**`sample_path`**(*times*, *events*, *states*, *model*, *time_start*, *time_end*, *color_palette=None*, *labelsize=16*, *ticksize=14*, *legendsize=16*, *num=1000*, *s=12*, *savefig=False*, *path=”*, *fig_name=’sample_path.pdf’*)

Plots a sample path along with the intensities.

Parameters

- **`times`** (*`array of floats`*) – times when the events occur.
- **`events`** (*`array of int`*) – type of the event at each event time.
- **`states`** (*`array of int`*) – state process after each event time.
- **`model`** (*`HybridHawkesExp`*) – the model that is used to compute the intensities.
- **`time_start`** (*`float`*) – time at which the plot starts.
- **`time_end`** (*`float`*) – time at which the plot ends.
- **`color_palette`** (*`list of colours`*) – one colour per event type.
- **`labelsize`** (*`int`*) – fontsize of labels.
- **`ticksize`** (*`int`*) – fontsize of tick labels.
- **`legendsize`** (*`int`*) – fontsize of the legend.
- **`num`** (*`int`*) – number of points used to plot.
- **`s`** (*`int`*) – size of the dots in the scatter plot of the events.
- **`savefig`** (*`boolean`*) – set to True to save the figure.
- **`path`** (*`string`*) – where the figure is saved.
- **`fig_name`** (*`string`*) – name of the file.

Return type Figure, array of Axes

Returns the figure and array of figures (see matplotlib).

`mpoints.plot_tools.`**`transition_probabilities`**(*probabilities, shape=None, path='', fig_name='transition_probabilities.pdf', events_labels=None, states_labels=None, title=None, color_map=None, figsize=(12, 6), size_labels=16, size_values=14, bottom=0.1, top=0.95, left=0.08, right=0.92, wspace=0.2, hspace=0.2, savefig=False, usetex=False*)

Annotated heatmap of the transition probabilities of a state-dependent Hawkes process.

Parameters

- **`probabilities`** (*`3D array`*) – the transition probabilities.
- **`shape`** (*`(int, int)`*) – 2D-tuple (number of rows, number of columns), shape of the array of figures.
- **`path`** (*`string`*) – where the figure is saved.
- **`fig_name`** (*`string`*) – name of the file.
- **`events_labels`** (*`list of strings`*) – labels of the event types.
- **`states_labels`** (*`list of strings`*) – labels of the states.
- **`title`** (*`string`*) – suptitle.
- **`color_map`** – color map for the heatmap, see seaborn documentation.
- **`figsize`** (*`(int, int)`*) – tuple (width, height).
- **`size_labels`** (*`int`*) – fontsize of labels.
- **`size_values`** (*`int`*) – fontsize of the annotations on top of the heatmap.
- **`bottom`** (*`float`*) – between 0 and 1, adjusts the bottom margin, see matplotlib subplots_adjust.
- **`top`** (*`float`*) – between 0 and 1, adjusts the top margin, see matplotlib subplots_adjust.
- **`left`** (*`float`*) – between 0 and 1, adjusts the left margin, see matplotlib subplots_adjust.
- **`right`** (*`float`*) – between 0 and 1, adjusts the right margin, see matplotlib subplots_adjust.
- **`wspace`** (*`float`*) – horizontal spacing between the subplots, see matplotlib subplots_adjust.
- **`hspace`** (*`float`*) – vertical spacing between the subplots, see matplotlib subplots_adjust.
- **`savefig`** (*`boolean`*) – set to True to save the figure.
- **`usetex`** (*`boolean`*) – set to True if matplolib figure is rendered with TeX.

Return type Figure, array of Axes

Returns the figure and array of figures (see matplotlib).

CHAPTER 5

References

The *mpoints* package is based on the following two references. The first reference is an introduction to state-dependent Hawkes processes with an application to high-frequency financial data. The second reference sets the theoretical foundations of hybrid marked point processes.

1. Morariu-Patrichi, M. and Pakkanen, M. S. (2018). State-dependent Hawkes processes and their application to limit order book modelling. Preprint, available at http://arxiv.org/abs/1809.08060.
2. Morariu-Patrichi, M. and Pakkanen, M. S. (2017). Hybrid marked point processes: characterisation, existence and uniqueness. Preprint, available at: http://arxiv.org/abs/1707.06970.

A friendly introduction to (hybrid) marked point processes can be found at https://maximemorariu.com.

CHAPTER 6

Support

Support can be found on the GitHub page of the project. Should you have any queries, feel free to contact Maxime Morariu-Patrichi via the contact form on his website.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index