On the need for latency as a parameter in exponential and sum-of-exponentials in Tick

Marcos Costa Santos Carreira

Jul-2020

1 Setup

1.1 Imports

Algorithm 1 Imports

```
import numpy as np
import pandas as pd
from tick.hawkes import SimuHawkes, SimuHawkesMulti
from tick.base import TimeFunction
from tick.hawkes import HawkesKernelTimeFunc
from tick.hawkes import HawkesKernelSumExp
from tick.hawkes import HawkesEM, HawkesSumExpKern
from tick.plot import plot_timefunction
from tick.plot import plot_point_process
from tick.plot import plot_hawkes_kernels
from tick.plot import plot_basis_kernels
```

1.2 Constants and functions

A small latency (kernel discretized with enough resolution to pick it up):

Algorithm 2 Constants and functions

```
support = 4
t0 = 0.01
n_steps = int(support/t0)*10

def g1(t):
        return 0.7 * 5.0 * np.exp(-5.0 * t)

def g2(t):
    return 0.7 * 5.0 * np.exp(-5.0 * (t - t0))\
        * np.heaviside(t - t0, 1) # To ensure zero before t0

def time_func(f, support, t0=0, steps=1000):
    t_values = np.linspace(0, support, steps + 1)
    y_values = f(t_values - t0) * np.heaviside(t_values - t0, 1)
    return TimeFunction(values=(t_values, y_values),
                        border_type=TimeFunction.Border0,
                        inter_mode=TimeFunction.InterLinear)

tf_1 = time_func(g1, support, 0, n_steps)
tf_2 = time_func(g1, support, t0, n_steps)
```

1.3 Time function plots

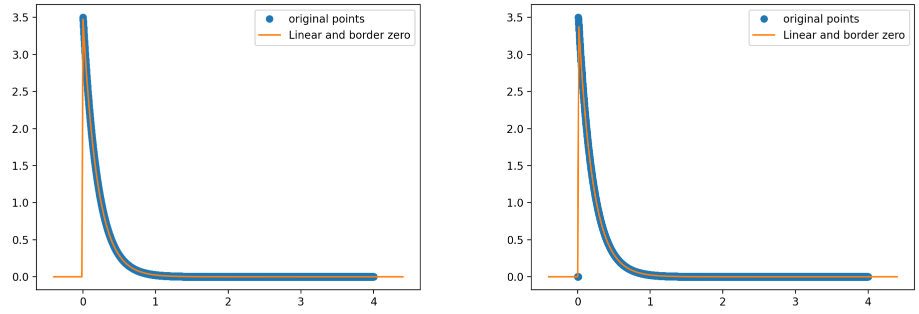

Figure 1: Time functions

2 Simulations

Algorithm 3 Simulation

```
kernel_1 = HawkesKernelTimeFunc(tf_1)
kernel_2 = HawkesKernelTimeFunc(tf_2)

hawkes_m1 = SimuHawkes(n_nodes=1, end_time=10000)
hawkes_m1.set_baseline(0, 1.)
hawkes_m2 = SimuHawkes(n_nodes=1, end_time=10000)
hawkes_m2.set_baseline(0, 1.)

hawkes_m1.set_kernel(0, 0, kernel_1)
hawkes_m2.set_kernel(0, 0, kernel_2)

multi_1 = SimuHawkesMulti(hawkes_m1, n_simulations=100)
multi_1.simulate()
multi_1_timestamps = multi_1.timestamps

multi_2 = SimuHawkesMulti(hawkes_m2, n_simulations=100)
multi_2.simulate()
multi_2_timestamps = multi_2.timestamps
```

3 Learners

3.1 Non-parametric

3.1.1 HawkesEM

Running HawkesEM for the 2 kernels:

Algorithm 4 HawkesEM

```
kern_d =\
    np.concatenate(
        (np.array([0., 0.2*t0,  0.5*t0, 0.75*t0, 0.9*t0, t0, 1.25*t0, 1.5*t0]),
         np.array([0.02, 0.05, 0.075, 0.1, 0.2, 0.5, 0.75, 1., 2.,
                   support])))

em_1 = HawkesEM(kernel_discretization=kern_d, max_iter=10000, tol=1e-5,
                verbose=True, n_threads=-1)
em_1.fit(multi_1_timestamps)
em_1_kernel = em_1.kernel

em_2 = HawkesEM(kernel_discretization=kern_d, max_iter=10000, tol=1e-5,
                verbose=True, n_threads=-1)
em_2.fit(multi_2_timestamps)
em_2_kernel = em_2.kernel
```

Differences in the learned kernels:

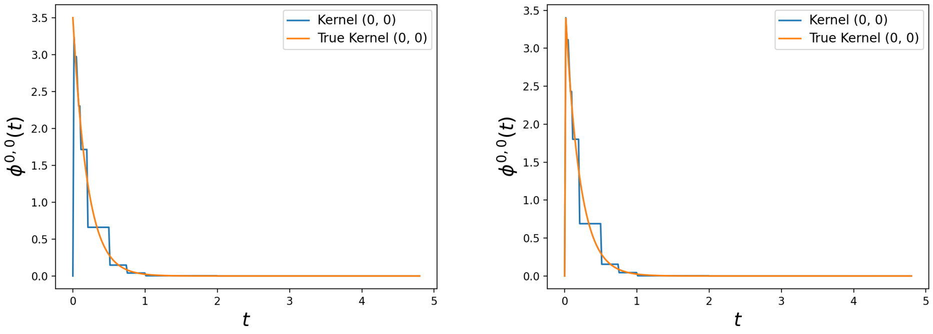

Figure 2: Learned kernels HawkesEM; left: no latency; right: latency=0.01

It is better to look at the log plots:

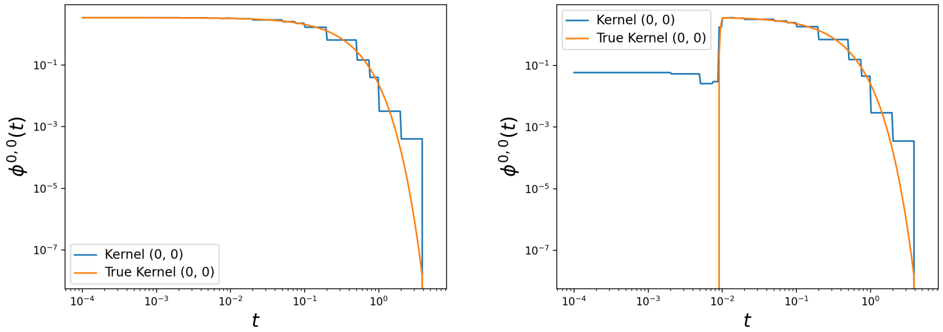

Figure 3: Learned kernels HawkesEM - LogPlot; left: no latency; right: latency=0.01

3.2 Parametric

3.2.1 HawkesSumExpKern

Running HawkesSumExpKern for the 2 kernels:

Algorithm 5 HawkesSumExp

```
sek_1 = HawkesSumExpKern(decays=[5.],
                         n_baselines=1, penalty='l2', solver='agd',
                         elastic_net_ratio=0.8,
                                                 max_iter=10000, tol=1e-5, C=100
sek_1.fit(multi_1_timestamps)
sek_1_baseline = sek_1.baseline
sek_1_adjacency = sek_1.adjacency

sek_2 = HawkesSumExpKern(decays=[5.],
                         n_baselines=1, penalty='l2', solver='agd',
                         elastic_net_ratio=0.8,
                         max_iter=10000, tol=1e-5, C=1000., verbose=True)
sek_2.fit(multi_2_timestamps)
sek_2_baseline = sek_2.baseline
sek_2_adjacency = sek_2.adjacency
```

Differences in the learned kernels:

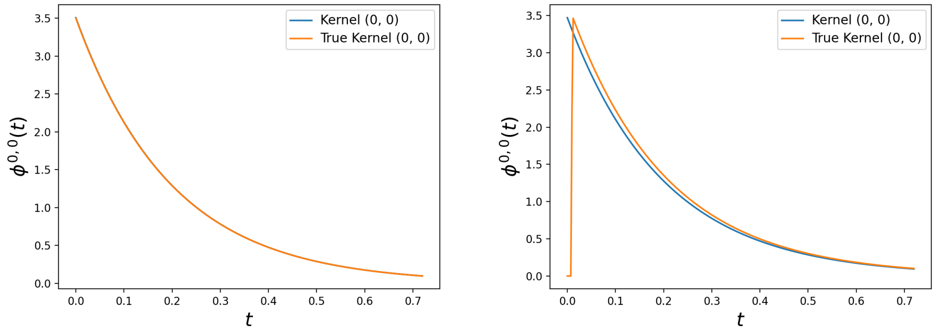

Figure 4: Learned kernels HawkesSumExpKern; left: no latency; right: latency=0.01

It is better to look at the log plots:

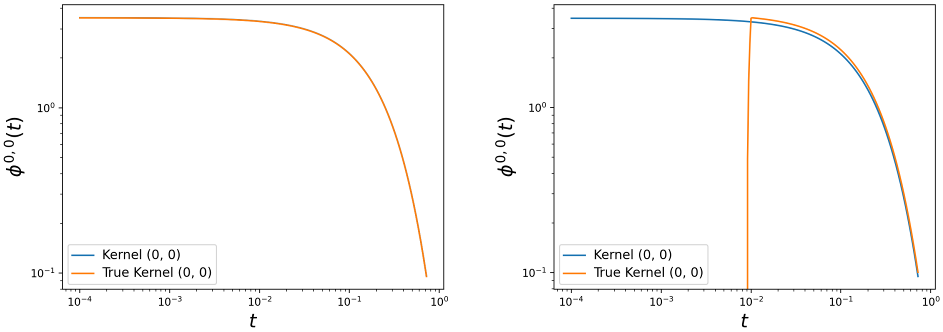

Figure 5: Learned kernels HawkesEM - LogPlot; left: no latency; right: latency=0.01

4 Conclusions

In the same way that decay is an input to HawkesSumExpKern, latency should be an input, especially for financial data. It would be much better to calibrate the parametric learner considering this input instead of using HawkesEM, since the simulation with HawkesKernelSumExp would also benefit from it. And even allowing for an array as an input (same sas decay) is good, because different reactions might have different latencies as well.