

# tests\_comparison

May 6, 2024

## 1 Comparison between different tests

In this document we will show some different attempts to produce goodness-of-fit tests for point processes. We will begin with our bootstrap-based procedure and then show other non-bootstrap-based ones.

The procedures described in [Bootstrapping interarrival times](#) and [No simulation - uniform distribution](#) were introduced in [Ogata \(1988\)](#) and are still used in the literature.

Some of the methods presented do perform well, especially for Poisson processes, but for Hawkes processes with parameters estimated from the data we are working with, the bootstrap-based approach provided a more consistent goodness-of-fit test.

For a reference on Point processes and the results mentioned here, the reader can refer to [Laub \(2021\)](#).

```
[1]: using PointProcessTools
      using CSV
      using DataFrames
      using Plots
      import Distributions: Uniform, Exponential, mean, cdf
      using Random
      import HypothesisTests: ExactOneSampleKSTest, pvalue

[ ]: ## Load the data
rec = Record("./data/record.csv", GeologicalSetting = ["Volcanic Front",
    ↪ "Active Rift"])
pars_hp = PointProcessTools.estimate("hp", rec) # Parameter for the Poisson
    ↪ process estimated from data
pars_hh = PointProcessTools.estimate("hh", rec) # Parameter for the Hawkes
    ↪ process estimated from data
n_tests = 5000; # Number of p-values generated for each test
N = 1000; # 'N' from step 2 in the following algorithms
```

### 1.1 How the tests are performed

In the following, we will test the different procedures to determine if they are consistent goodness-of-fit tests.

Given some observed data (an eruption record, in our case), a goodness-of-fit test will provide a p-value for each hypothesis on how the data is distributed (example, it might return a p-value of 34% for the hypothesis that the data is Poisson, or 12% for the hypothesis of Hawkes process).

A consistent test, when tested on data stemming from the correct hypothesis, should produce uniformly distributed p-values. In our specific case, what will be done is that we will simulate  $K$  Poisson (Hawkes) processes and, for each of these simulations, calculate the p-value for the hypothesis that the data is from a Poisson (Hawkes) process. If the test is consistent, these  $K$  p-values should be uniformly distributed.

The simulation will use parameters estimated from the data used in the paper, as to provide an example based on real data.

## 1.2 Bootstrap procedure

Our procedure is a bootstrap based goodness-of-fit test. Assume we have an event record and we want to test if it is a Poisson (or any other) process. The algorithm goes as follows:

1. Estimate the parameter of the process from the data and calculate its KS-distance  $d$  using this estimated parameter.
2. Simulate  $N$  Poisson process using the estimated parameter.
3. Estimate the parameters of each of the  $N$  simulations.
4. Calculate the KS-distances  $d_n$  for each of the  $N$  simulated process with its corresponding estimated parameter.
5. The p-value of the test is the proportion of distances  $d_n$  that are larger than the KS-distance  $d$  calculated from the original process.

```
[6]: # Algorithm of the bootstrap based test
function bootstrap_based(data, model, N)
    data_pars = estimate(model, data) # Estimate parameters as in step 1
    d = distance("ks", time_transform(data_pars, data.events, data.interval))
    ↪ # Calculates 'd' in step 1
    dn = zeros(N) # Where the dn's from step 4 will be stored
    for i in 1:N
        sim = simulate(data_pars, data) # Step 2
        sim_pars = estimate(PointProcessTools.str2model(model), sim, data.T) #
    ↪ Step 3
        dn[i] = distance("ks", sim_pars, sim, data.interval) # Step 4
    end
    return sum(dn .>= d) / N # Step 5
end
```

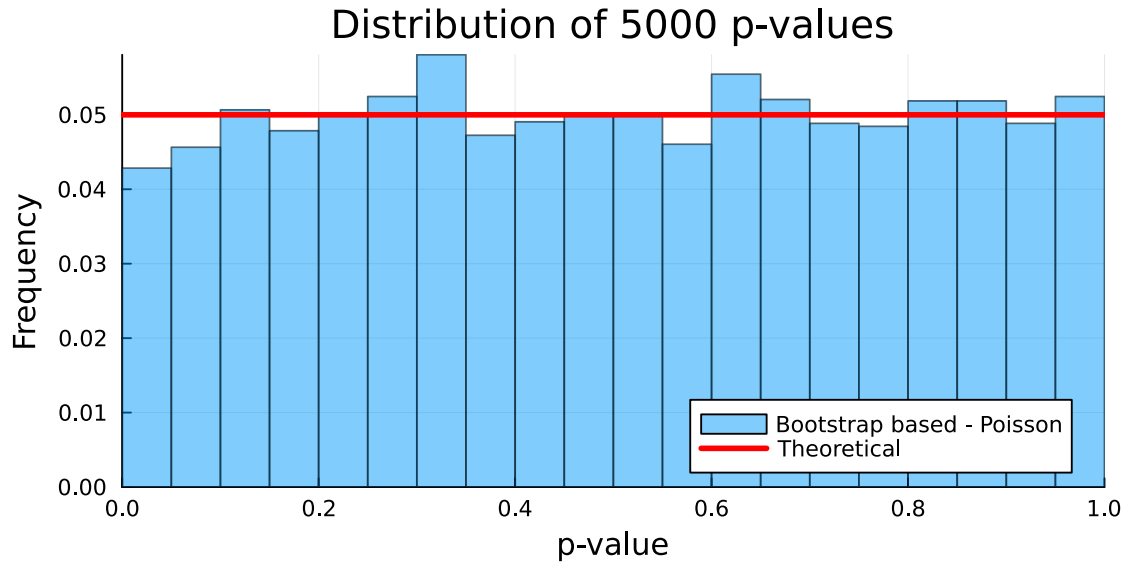
bootstrap\_based (generic function with 1 method)

This is the performance of our bootstrap based goodness-of-fit test for Poisson processes.

Notice how the p-values are consistent with a uniform distribution, which is exactly how a consistent test should perform.

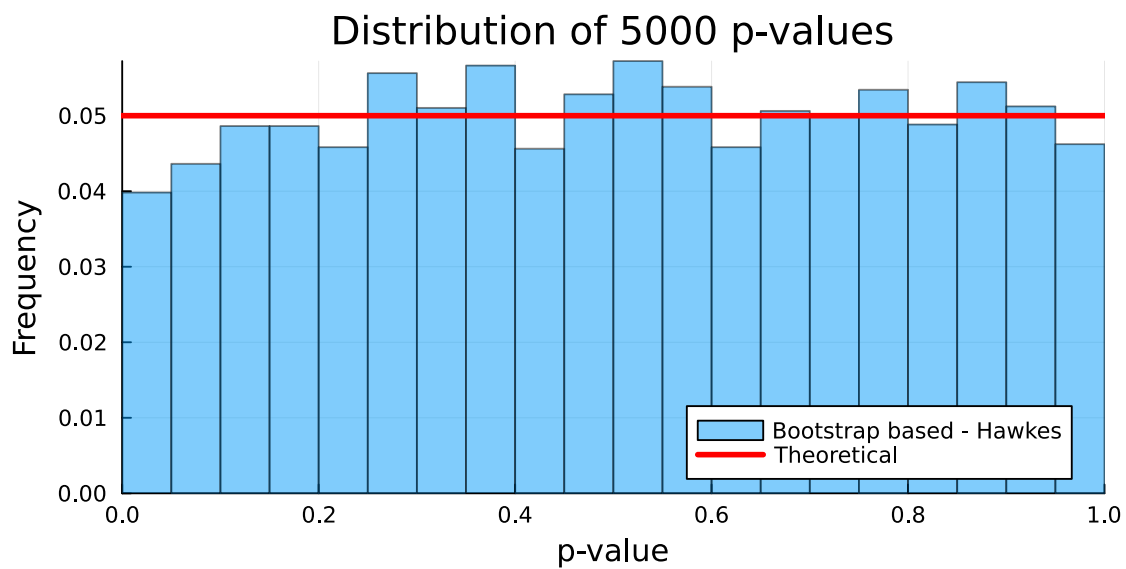
```
[24]: model = "hp" # Testing for Poisson process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hp, rec), rec.interval)
    ps[i] = bootstrap_based(data, model, N)
end
```

```
plot_ps(ps, "Bootstrap based - Poisson")
```



This is the performance for Hawkes processes.

```
[41]: model = "hh" # Testing for Hawkes process
      ps = zeros(n_tests)
      for i in 1:n_tests # Repeat the algorithm 'n_tests' times
          data = Record(simulate(pars_hh, rec), rec.interval)
          ps[i] = bootstrap_based(data, model, N)
      end
      plot_ps(ps, "Bootstrap based - Hawkes")
```



### 1.3 Test without bootstrapping step

This example is the same as presented in Section 4.1 of the paper. The algorithm is:

1. Estimate the parameter of the process from the data and calculate its KS distance  $d$  using this estimated parameter.
2. Simulate  $N$  Poisson process using the estimated parameter.
3. Calculate the KS distances  $d_n$  for each of the  $N$  simulated process with estimated parameter from step 1.
4. The p-value of the test is the proportion of distances  $d_n$  that are larger than the KS-distance  $d$  calculated from the original process.

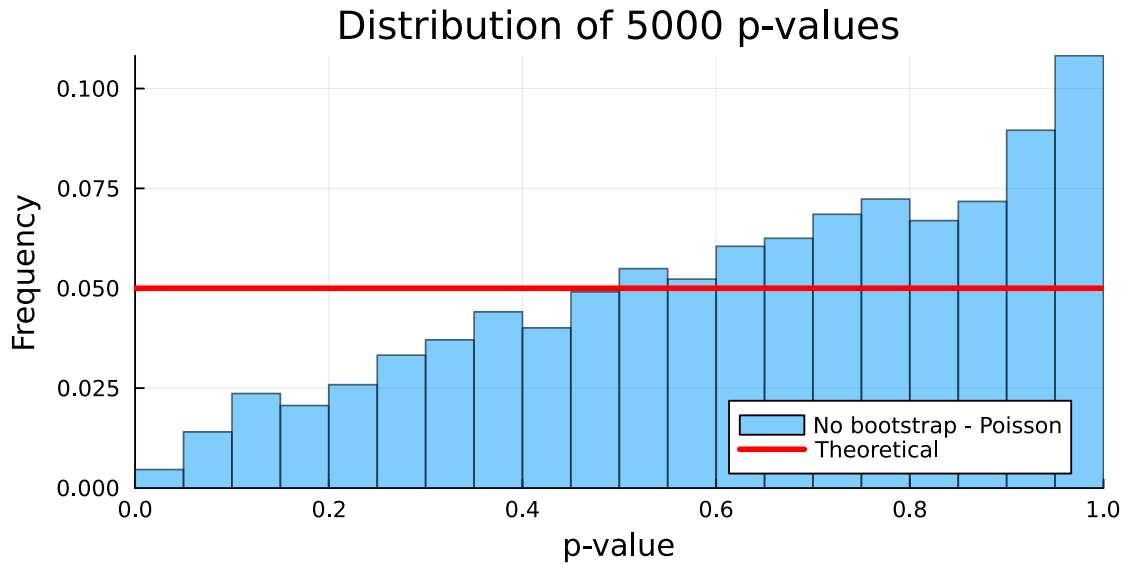
Notice that step 3 from the previous algorithm is skipped in this algorithm, therefore, this test does not account for possible errors made when estimating the parameters.

```
[6]: # Algorithm for test without bootstrap step
function fixed_exponential(data, model, N)
    data_pars = estimate(model, data) # Estimate parameters as in step 1
    d = distance("ks", time_transform(data_pars, data.events, data.interval))
    ↪ # Calculates 'd' in step 1
    dn = zeros(N) # Where the dn's from step 3 will be stored
    for i in 1:N
        sim = simulate(data_pars, data) # Step 2
        dn[i] = distance("ks", time_transform(data_pars, sim, data.interval)) #
    ↪ Step 3. Notice that the parameters from 'sim' were not estimated.
    end
    return sum(dn .>= d) / N # Step 4
end
```

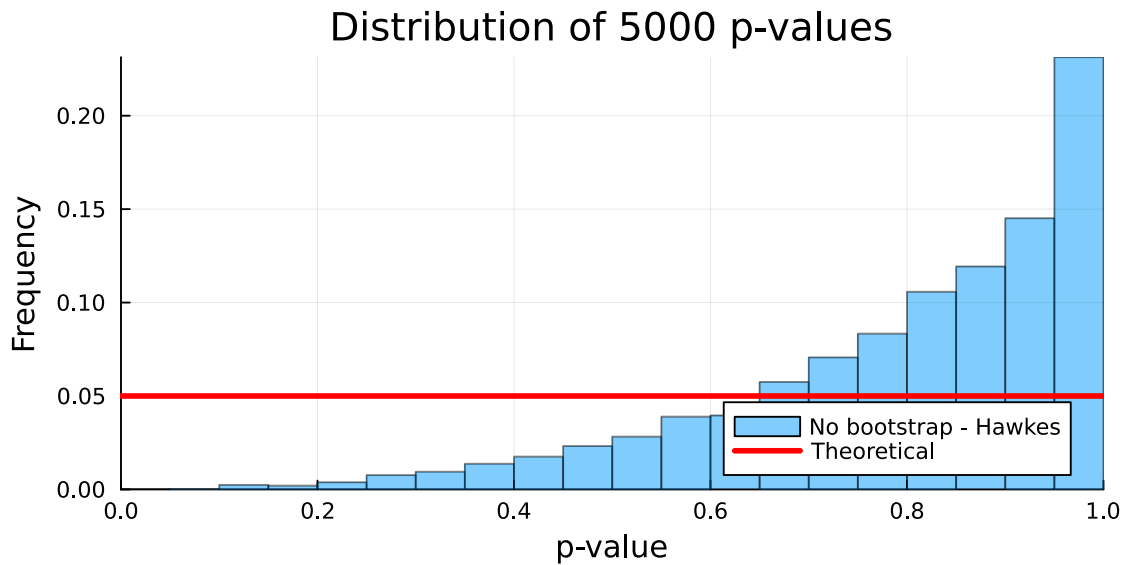
fixed\_exponential (generic function with 1 method)

The same tests as above were carried out for the non-bootstrap based algorithm. We first display the results for the Poisson case and then for the Hawkes case.

```
[25]: model = "hp" # Testing for Poisson process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hp, rec), rec.interval) # Represents real data
    ps[i] = fixed_exponential(data, model, N)
end
plot_ps(ps, "No bootstrap - Poisson")
```



```
[26]: model = "hh" # Testing for Hawkes process
      ps = zeros(n_tests)
      for i in 1:n_tests # Repeat the algorithm 'n_tests' times
          data = Record(simulate(pars_hh, rec), rec.interval) # Represents real data
          ps[i] = fixed_exponential(data, model, N)
      end
      plot_ps(ps, "No bootstrap - Hawkes")
```



It is clear by the two plots that this does not produce a consistent goodness-of-fit test. Both algorithms are tested the exact same way. In each iteration, we simulate a process from a known

model and get the p-value of both tests. A consistent goodness-of-fit test, in this situation, should produce a distribution of p-values close to uniform.

## 1.4 No simulation

This algorithm simply performs the Kolmogorov-Smirnov test on the time transformed interarrival times against a unit exponential distribution. Since there are tables available to calculate the p-values based on the calculated distance, this would be the most efficient algorithm.

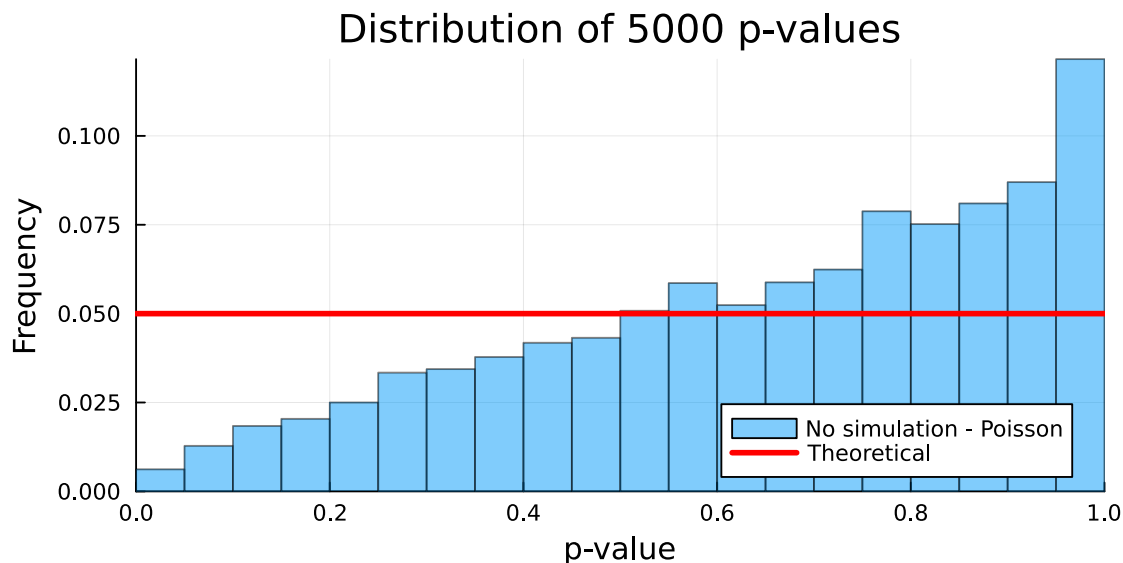
1. Estimate the parameter of the process from the data.
2. Calculate the time transformed interarrival times.
3. Perform the KS-test against the unit exponential distribution.

```
[7]: # No simulation algorithm
function no_simulation(data, model, N)
    data_pars = estimate(model, data) # Step 1
    interarrivals = diff(time_transform(data_pars, data.events, data.interval))
    ↪ # Step 2
    p = pvalue(ExactOneSampleKSTest(interarrivals, Exponential(1))) # Step 3
    return p
end
```

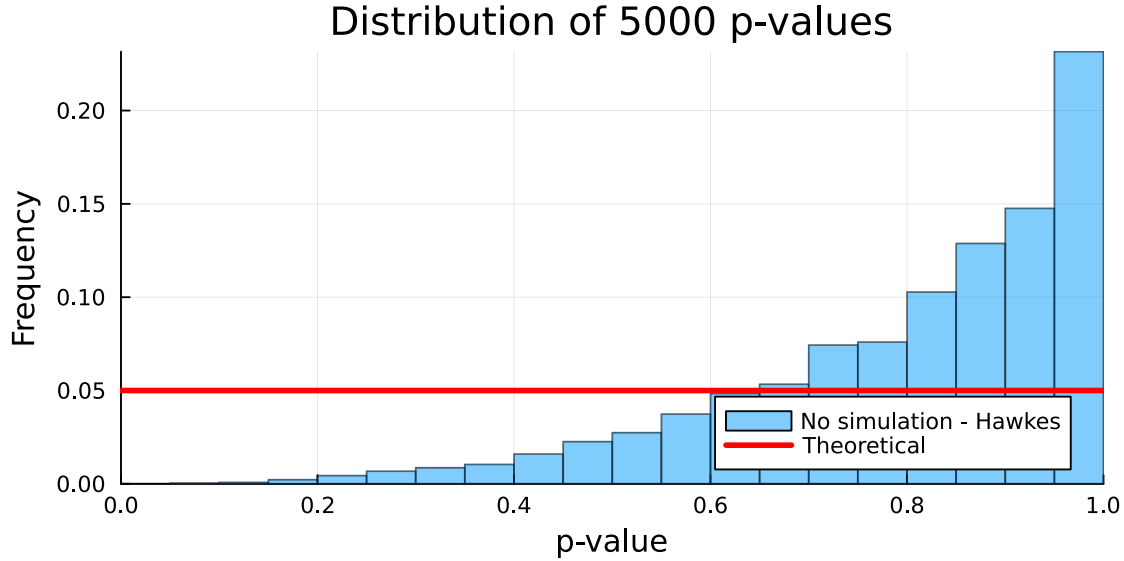
no\_simulation (generic function with 1 method)

This procedure performs similarly to the previous one.

```
[8]: model = "hp" # Testing for Poisson process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hp, rec), rec.interval) # Represents real data
    ps[i] = no_simulation(data, model, N)
end
plot_ps(ps, "No simulation - Poisson")
```



```
[9]: model = "hh" # Testing for Hawkes process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hh, rec), rec.interval) # Represents real data
    ps[i] = no_simulation(data, model, N)
end
plot_ps(ps, "No simulation - Hawkes")
```



## 1.5 Bootstrapping interarrival times

This method aims to indirectly account for the estimation error of the previous procedure. The strategy here is to, instead of simulating  $N$  point processes in step 2 from the bootstrap algorithm, simulate  $N$  sets of interarrival times. This would simplify the problem, since estimating the parameter of an exponential distribution is simpler than estimating parameters of a Hawkes process. This procedure is equivalent to applying packages such as R's [KScorrect](#) to the transformed interarrival times against an exponential distribution.

Notice, however, that this does not fully reflect the variation present in point processes. First, it is not clear whether the error in parameter estimation is being considered, since it is the exponential distribution's parameters that are being estimated, not the process' parameters. Second, realizations of the same point process can have different number of events, but in this case, since the simulation is done with respect to the interarrival times, we are simulating always a fixed number of events.

Here is the algorithm:

1. Estimate the parameter of the process from the data and calculate its KS-distance  $d$  using this estimated parameter.

2. Compute the interarrival times of the time transformed process using the estimated parameter. Assume there are  $I$  interarrival times.
3. Estimate the distribution of the interarrival times in the previous step as an exponential distribution.
4. Simulate  $N$  sets of  $I$  interarrival times with the distribution estimated in step 3.
5. For each of the  $N$  sets, estimate the parameter as an exponential distribution and calculate the KS-distances  $d_n$  with respect to the corresponding estimated parameter.
6. The p-value is given by  $(\#(d_n > d) + 1)/(N + 1)$ .

```
[9]: # Algorithm using KScorrect
function KScorrect(data, model, N)
    data_pars = estimate(model, data) # Estimate parameters as in step 1
    interarrivals = diff(time_transform(data_pars, data.events, data.interval))
    ↪ # Step 2
    d = KSdist_exponential(interarrivals) # Calculates 'd' in step 1
    distribution = Exponential(mean(interarrivals)) # Step 3
    dn = zeros(N) # Where the dn's from step 4 will be stored
    for i in 1:N
        sim_interarrivals = rand(distribution, length(interarrivals)) # Step 4
        dn[i] = KSdist_exponential(sim_interarrivals) # Step 5
    end
    return (sum(dn .>= d) + 1) / (N + 1) # Step 6
end
```

KSdist\_exponential (generic function with 1 method)

```
[ ]: # KS-distance for an exponential distribution
function KSdist_exponential(x::Vector{Float64})
    l = length(x)
    sort!(x)
    # 'mean(x)' is the estimated parameter of the exponential distribution.
    d = Exponential(mean(x))
    dist_lower = abs.(cdf(d, x) .- ((0:l-1)/l))
    dist_upper = abs.(cdf(d, x) .- ((1:l)/l))
    return maximum([dist_lower; dist_upper])
end
```

Below is the performance of the test applied to a Poisson process.

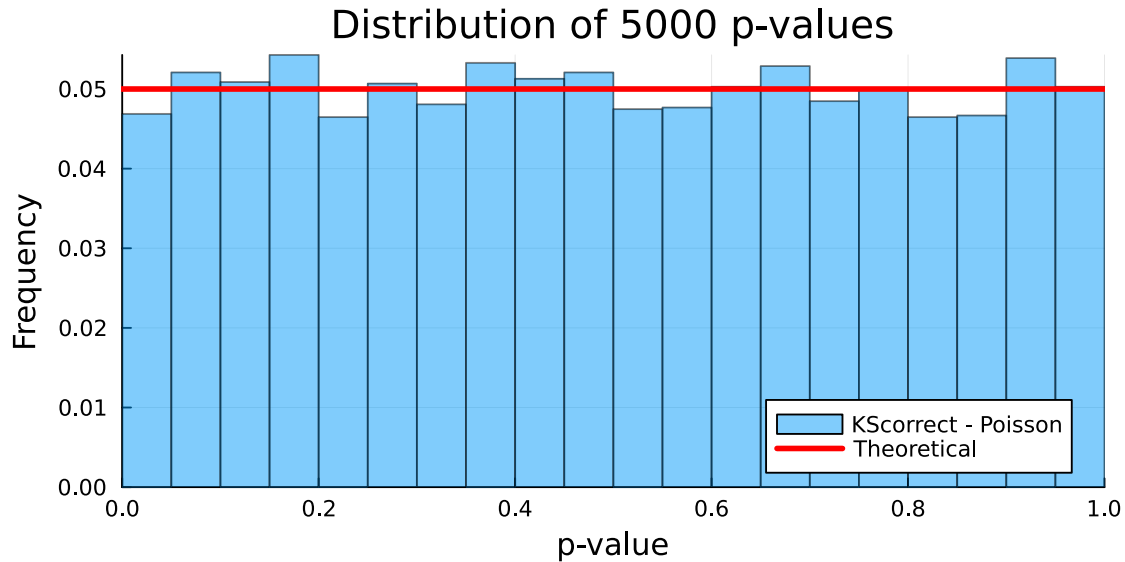
```
[27]: model = "hp" # Testing for Poisson process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hp, rec), rec.interval) # Represents real data
    ps[i] = KScorrect(data, model, N)
```



```

end
plot_ps(ps, "KScorrect - Poisson")

```

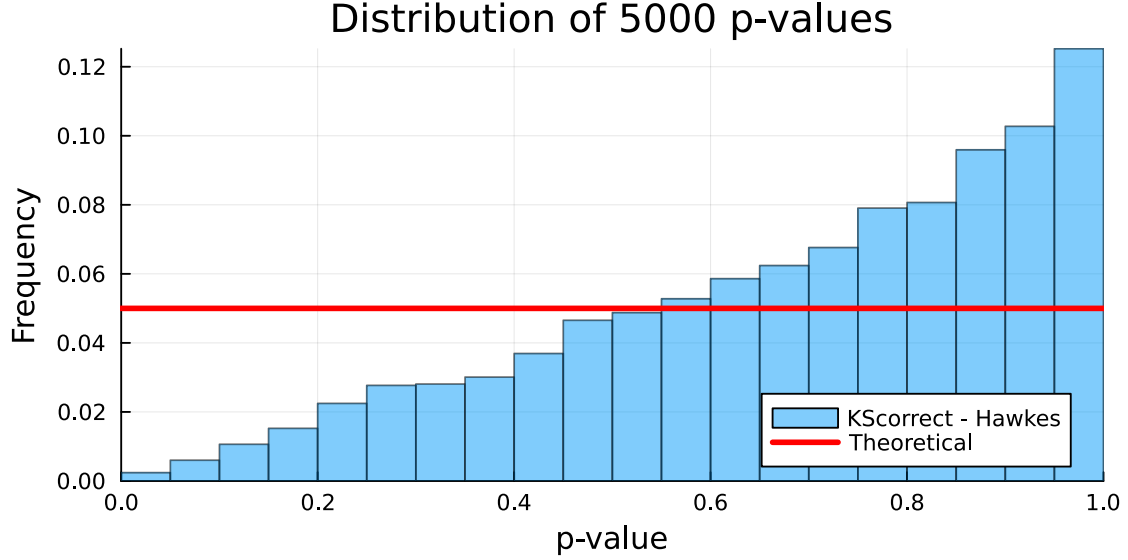


As we can see, this strategy is effective when testing for Poisson processes. However, for Hawkes processes, which are much more complex processes, the picture is different. The errors in estimating the parameters affect the distribution of the KS-distances in complex ways, which cannot be easily accounted for directly, therefore, we need to resort to simulations.

```

[28]: model = "hh" # Testing for Hawkes process
      ps = zeros(n_tests)
      for i in 1:n_tests # Repeat the algorithm 'n_tests' times
          data = Record(simulate(pars_hh, rec), rec.interval) # Represents real data
          ps[i] = KScorrect(data, model, N)
      end
      plot_ps(ps, "KScorrect - Hawkes")

```



## 1.6 Event times against uniform distribution

This variant is similar to the **no bootstrap procedure**, but instead of testing the transformed inter-arrival times against the exponential distribution, we will test the transformed event times against a uniform distribution.

1. Estimate the parameter of the process from the data and compute the time transformed event times using the estimated parameter.
2. Calculate  $d$  as the KS-distance of the transformed event times against a uniform distribution over  $[0, \int_0^T \lambda(t)dt]$ .
3. Simulate  $N$  point processes using the estimated parameter from 1 and calculate  $\int_0^T \lambda(t)dt$  (for Hawkes processes, this value depends on the event times).
4. Calculate the KS-distances  $d_n$  of the simulated event with respect to the uniform distribution in 3.
5. The p-value is given by  $(\#(d_n > d) + 1)/(N + 1)$ .

```
[8]: # Algorithm with event times against uniform distribution
function EventsUniform(data, model, N)
    data_pars = estimate(model, data) # Estimate parameters as in step 1
    transf_events = time_transform(data_pars, data.events, data.interval) #
    ↪ Step 1
    cif = CIF(data_pars, data) # Integral of the CIF
    d = KSdist_uniform(transf_events, cif) # Calculates 'd' in step 2
    dn = zeros(N) # Where the dn's from step 4 will be stored
    for i in 1:N
        sim_events = simulate(data_pars, data) # Step 3
        sim_cif = CIF(data_pars, sim_events, data.interval)
```

```

        dn[i] = KSdist_uniform(time_transform(data_pars, sim_events, data.
↪interval), sim cif) # Step 4
    end
    return (sum(dn .>= d) + 1) / (N + 1) # Step 5
end

```

EventsUniform (generic function with 1 method)

```

[6]: # Function for calculating the KS-distance with respect to an uniform
↪distribution
function KSdist_uniform(x::Vector{Float64}, T::Float64)
    l = length(x)
    sort!(x)
    dist_lower = abs.(((0:l-1)./l) .- (x./T))
    dist_upper = abs.(((1:l)./l) .- (x./T))
    return maximum([dist_lower; dist_upper])
end

```

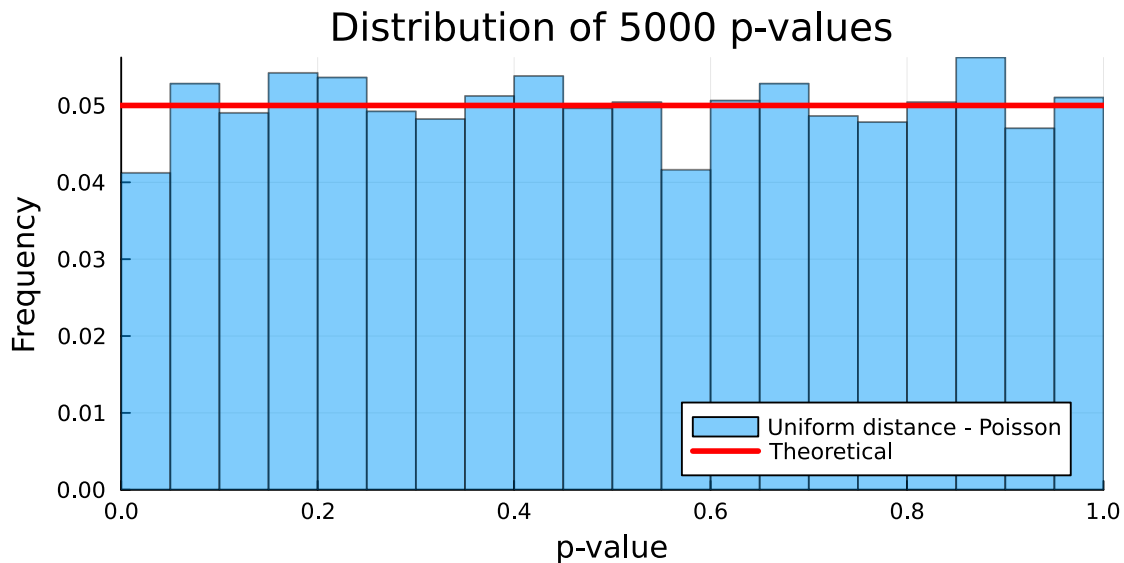
KSdist\_uniform (generic function with 1 method)

As we can see, this method works for Poisson Processes.

```

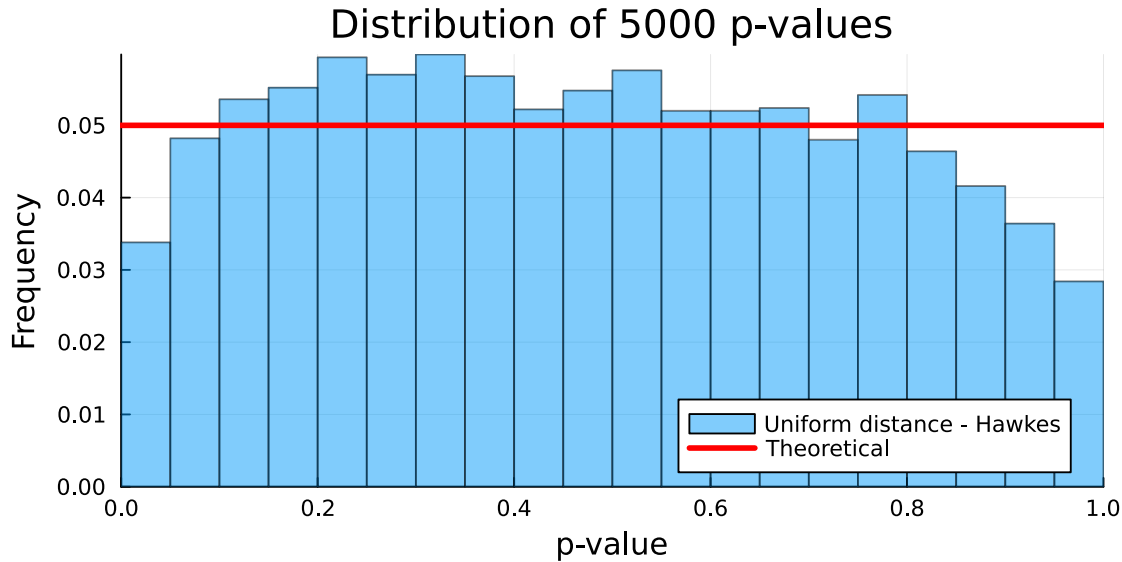
[9]: model = "hp" # Testing for Poisson process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hp, rec), rec.interval) # Represents real data
    ps[i] = EventsUniform(data, model, N)
end
plot_ps(ps, "Uniform distance - Poisson")

```



For the Hawkes case, there is a significant drop on both edges of the interval.

```
[10]: model = "hh" # Testing for Hawkes process
ps = zeros(n_tests)
for i in 1:n_tests # Repeat the algorithm 'n_tests' times
    data = Record(simulate(pars_hh, rec), rec.interval) # Represents real data
    ps[i] = EventsUniform(data, model, N)
end
plot_ps(ps, "Uniform distance - Hawkes")
```



## 1.7 No simulation - uniform distribution

It is well known that the event times of a Poisson process over an interval  $[0, T]$  are uniformly distributed. Also, if is  $N(t)$  is a point process with conditional intensity function  $\lambda$ , then the process constructed by performing the time transformation of  $N(t)$  with respect to  $\lambda$  is a Poisson process over the interval  $[0, \int_0^T \lambda(t)dt]$ , with intensity equal to 1.

The next test is similar to the procedure in the **No simulation** section. The only difference being that, instead of testing the transformed interarrival times against a unit exponential distribution, we test the transformed event times against a uniform distribution over  $[0, \int_0^T \lambda(t)dt]$ .

1. Estimate the parameter of the process from the data.
2. Calculate the time transformed event times.
3. Perform the KS-test against the uniform distribution over  $[0, \int_0^T \lambda(t)dt]$

```
[15]: # Algorithm simulating event times as uniform
function TestUniform(data, model, N)
    data_pars = estimate(model, data) # Step 1
    transf_events = time_transform(data_pars, data.events, data.interval) #
↪Step 2
```

```

    cif = CIF(data_pars, data) # Integral of the CIF
    distribution = Uniform(0, cif) # Construct the distribution in step 3
    p = pvalue(ExactOneSampleKSTest(transf_events, distribution)) # Step 3
    return p
end

```

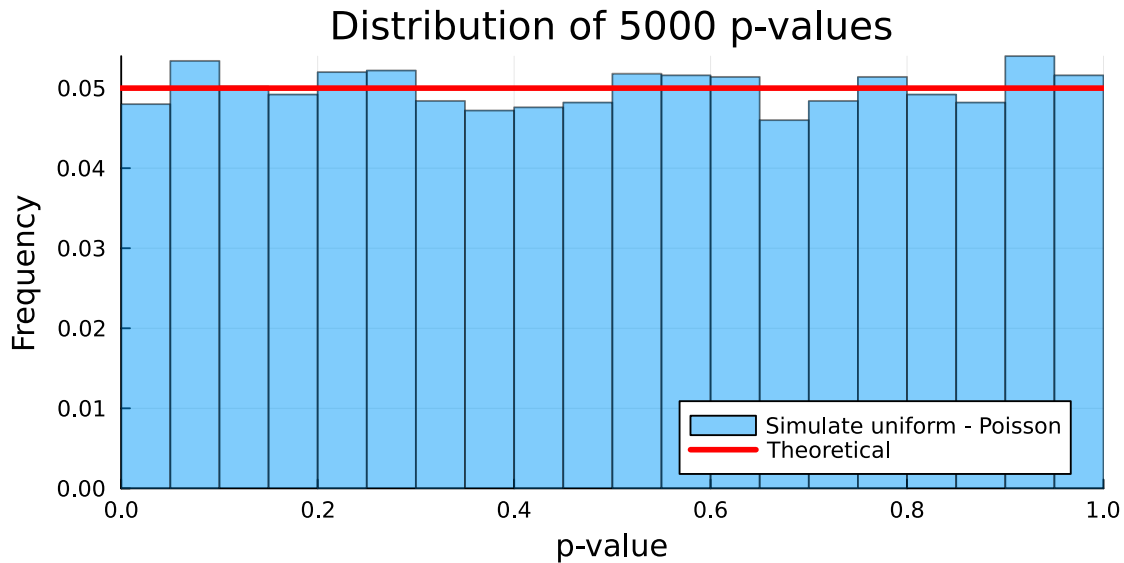
TestUniform (generic function with 1 method)

This method works well for Poisson processes, as can be seen below.

```

[16]: model = "hp" # Testing for Poisson process
      ps = zeros(n_tests)
      for i in 1:n_tests # Repeat the algorithm 'n_tests' times
          data = Record(simulate(pars_hp, rec), rec.interval) # Represents real data
          ps[i] = TestUniform(data, model, N)
      end
      plot_ps(ps, "Simulate uniform - Poisson")

```

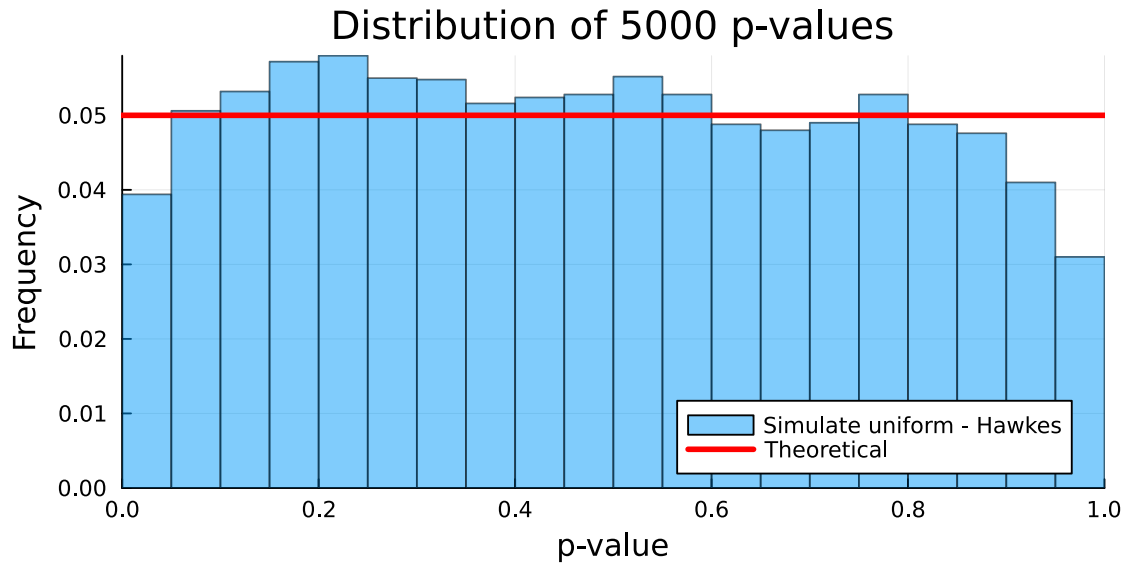


It is also the best performing of the non-bootstrap based models when applied to Hawkes processes. Although a drop is still present on the edges of the interval.

```

[17]: model = "hh" # Testing for Hawkes process
      ps = zeros(n_tests)
      for i in 1:n_tests # Repeat the algorithm 'n_tests' times
          data = Record(simulate(pars_hh, rec), rec.interval) # Represents real data
          ps[i] = TestUniform(data, model, N)
      end
      plot_ps(ps, "Simulate uniform - Hawkes")

```



```
[3]: function plot_ps(ps, label)
    p = histogram(ps, label=label, alpha=0.5, bins=0:0.05:1, normalize=:
    ↪probability)
    hline!(p, [0.05], label="Theoretical", c=:red, lw=3)
    plot!(p, xlabel="p-value", ylabel="Frequency", title="Distribution of
    ↪$(n_tests) p-values", xlim=(0, 1))
    plot!(p, size=(600, 300), left_margin=0.2Plots.cm, bottom_margin=0.2Plots.
    ↪cm, legend=:bottomright)
    display(p)
end

CIF(params::Parameters_hp, rec::Record) = params. * rec.T
CIF(params::Parameters_hp, _, interval::PointProcessTools.Interval) = params.
    ↪* length(interval)

CIF(params::Parameters_hh, rec::Record) = CIF(params, rec.events, rec.interval)
function CIF(params::Parameters_hh, events::Vector{Float64}, interval::
    ↪PointProcessTools.Interval)
    A = exp.(-params. .* (events .- interval.a))
    return (params. .* length(interval)) + sum((params. / params.) .-
    ↪((params. / params.) .* A))
end
```

CIF (generic function with 4 methods)