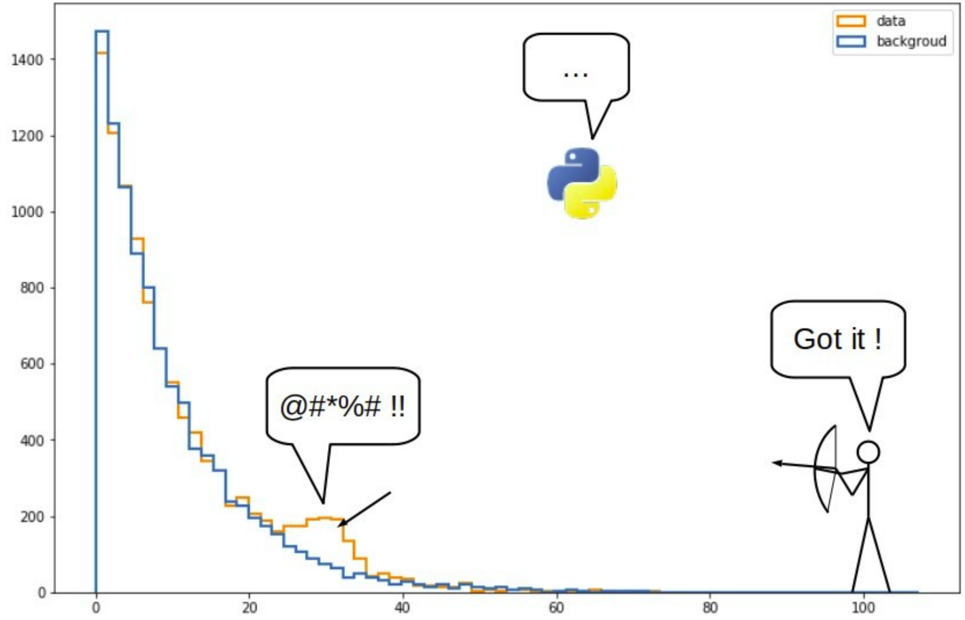
pyBumpHunter : A model agnostic bump hunting tool in python

Presenter :
Louis VASLIN – LPC Clermont



The BumpHunter algorithm

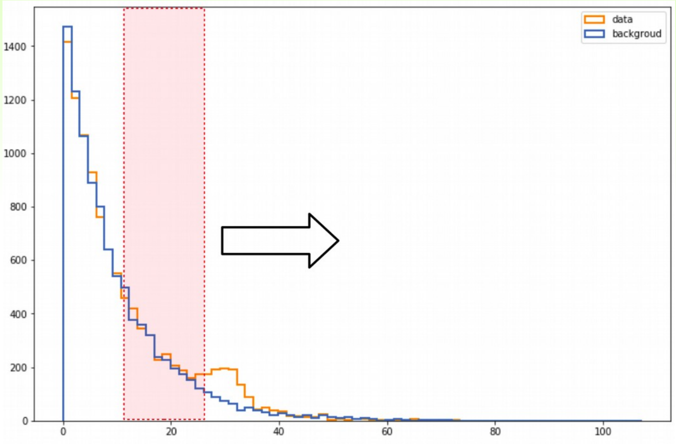
- Principle (based on arXiv:1101.0390v2)

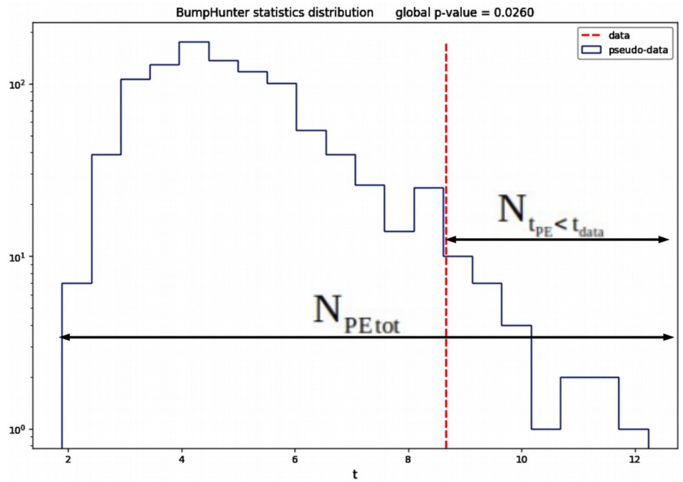
  Compare a data histogram with a reference background

  Test all intervals for various width and compute the **local p-value**
  => minimum p-value among m tests

  Generate **pseudo-data** by sampling the reference and repeat the scan process

  Infer the **global p-value** from the test statistic distribution of the pseudo-data (background-only)





The BumpHunter algorithm

- A bit of math

Local p-value (for a given interval)

$$\text{p-value} = \begin{cases} 1 - \Gamma(d+1, d) & \text{if } d \leq b \\ 1 & \text{otherwise} \end{cases}$$

For a deficit

$$\text{p-value} = \begin{cases} 1 & \text{if } d \leq b \\ \Gamma(d, b) & \text{otherwise} \end{cases}$$

For a excess

d = #data events
b = #background events

BumpHunter test statistic

$$t = -\ln(\text{p-value})$$

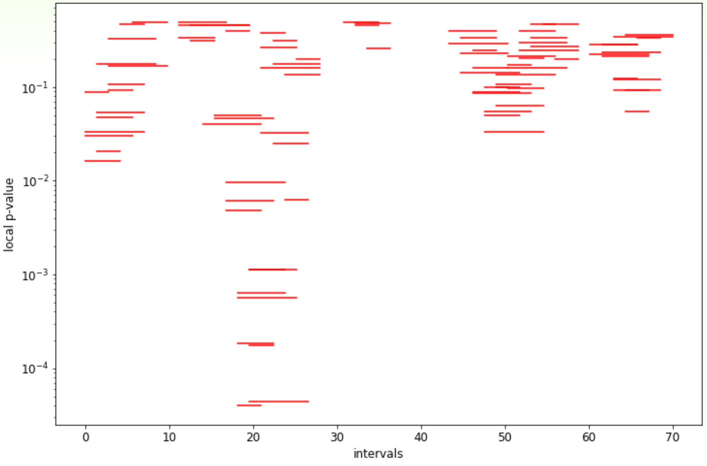
From p-value to significance

$$\text{sig} = \text{ppf}_{\text{normal}}(1-p)$$

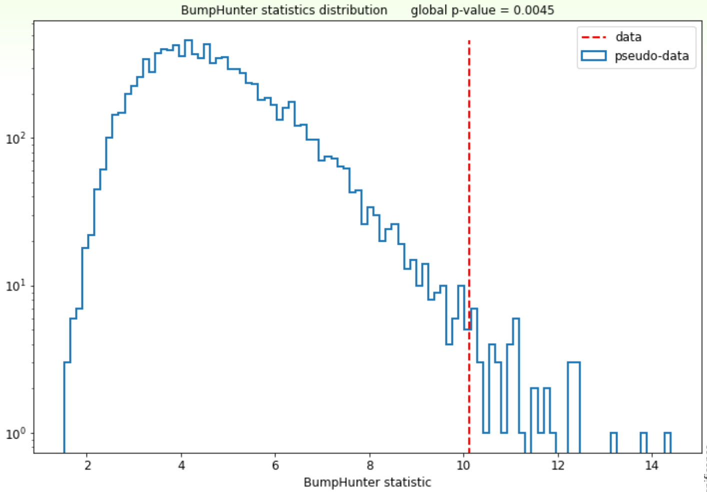
p = p-value
$\text{ppf}_{\text{normal}}$ = inverse cumulative function of a normal distribution

The BumpHunter algorithm

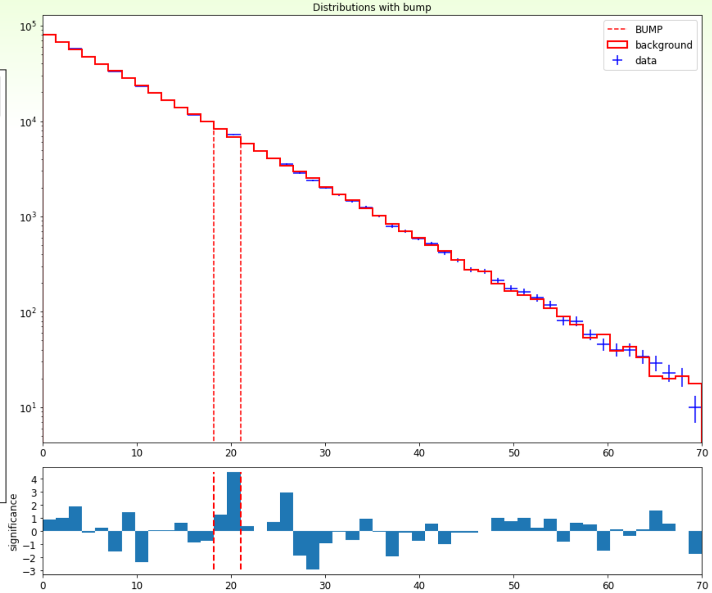
- Results



Tomography plot



BumpHunter test statistic



Bump plot

pyBumpHunter

- Public BumpHunter for python
  - **Pure python implementation** of the algorithm (github PyPI)
    - Depend only on numpy/scipy and matplotlib
    - pyBumpHunter is **pip installable**
  - Integrate **many extensions** of the base algorithm
    - Signal injection test
    - 2D BumpHunter
    - Automated side-band normalization
    - Multi-channel combination (next release)
  - pyBumpHunter has been **integrated in Scikit-HEP**
  - Development is **still ongoing**
    - Many new features will come

pyBumpHunter

- Simple example

Declare a BumpHunter1D instance

```
hunter = BH.BumpHunter1D(
    rang=rang,
    width_min=2,
    width_max=6,
    width_step=1,
    scan_step=1,
    npe=10000,
    nworker=1,
    seed=666,
)
```

Run a simple scan

```
hunter.bump_scan(data,bkg)
```

Produce the plots

```
hunter.plot_tomography(data)
```

```
hunter.plot_bump(data,bkg)
```

```
hunter.plot_stat(show_Pval=True)
```

Scan settings and results can be accessed through the BumpHunter class

For this configuration with 50 bins histograms, run time ~ 16s (on my laptop)

Jupyter notebook friendly

Signal injection and sensibility test

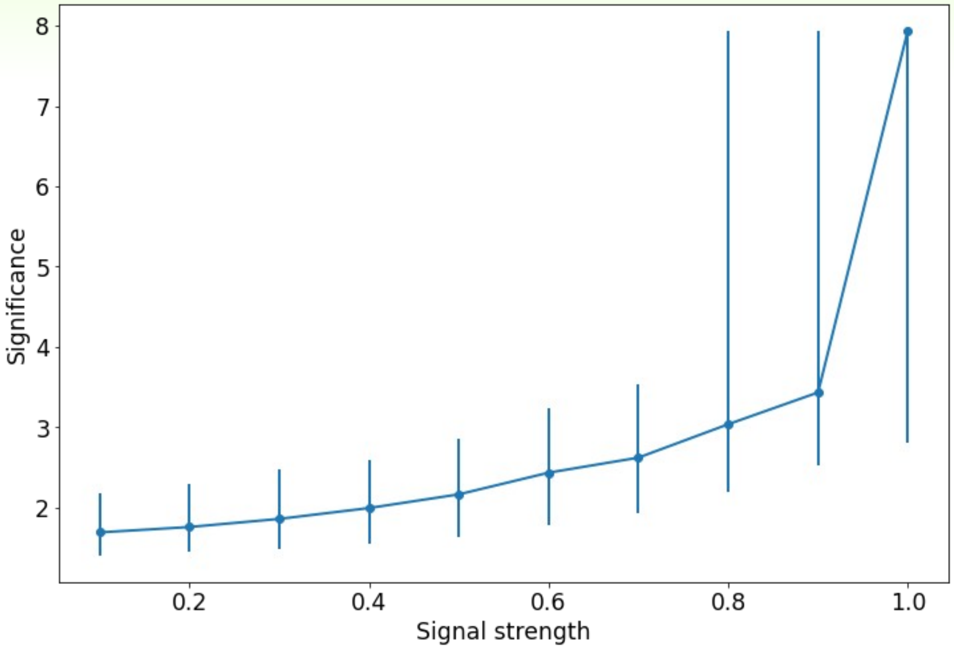
- Principle

Build a B+S pseudo-data histogram

Signal strength
=
#injected events / #expected

Apply BumpHunter algorithm on B+S pseudo-data

Increase signal strength until **required significance** is reached



Error bar obtained by producing **many** B+S histograms

Signal injection and sensibility test

- Code example

Additional injection settings

```
hunter.sigma_limit
hunter.str_min
hunter.str_scale
hunter.signal_exp
```

Run a signal injection test

```
hunter.signal_inject(sig,bkg)
```

Produce the plot

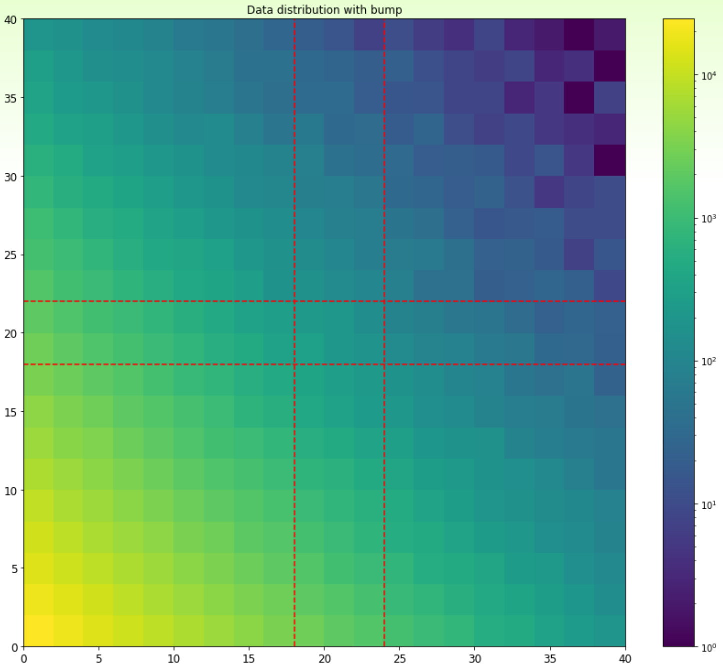
```
hunter.plot_inject()
```

Injection setting can be set when declaring the BumpHunter instance

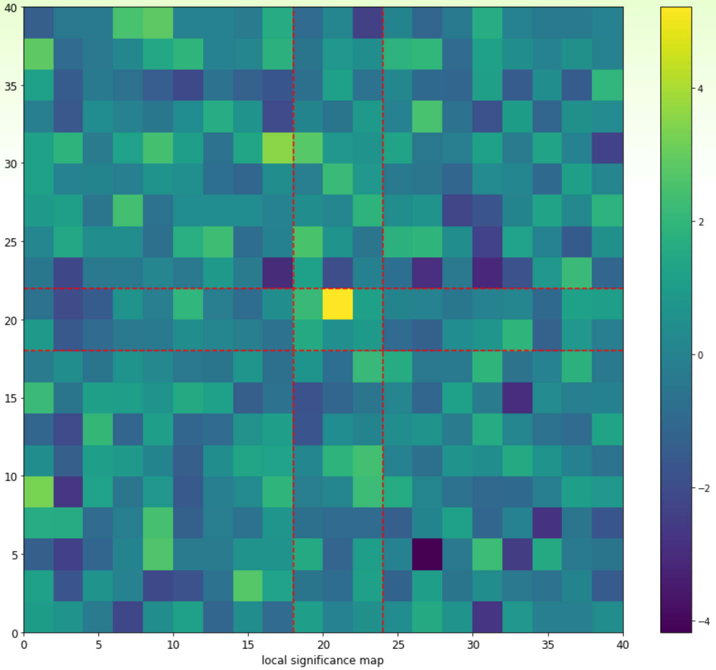
Injection plot produced both in linear and log scale (depend on str_scale setting)

With previous scan setting, run time ~3m30s

Bump hunting in 2D



2D data distribution



Local significance per bin

Same principle as in 1D

but **with 2D histograms**

Higher dimension
=
More sensible to statistics

<u>Only for basic scans</u>
(no 2D injection test yet)

Bump hunting in 2D

- Code example

Declare a BumpHunter2D instance

```
hunter = BH.BumpHunter2D(
    rang=rang,
    width_min=[2, 2],
    width_max=[3, 3],
    width_step=[1, 1],
    scan_step=[1, 1],
    bins=[20, 20],
    npe=8000,
    nworker=1,
    seed=666,
)
```

Run a simple scan

```
hunter.bump_scan(data,bkg)
```

Produce the plots

```
hunter.plot_bump(data,bkg)
```

```
hunter.plot_stat(show_Pval=True)
```

Same API as the BumpHunter1D class

For this configuration, run time ~1m30s (on my laptop)

Note : scan settings are now array-like of size 2

Side-band normalization

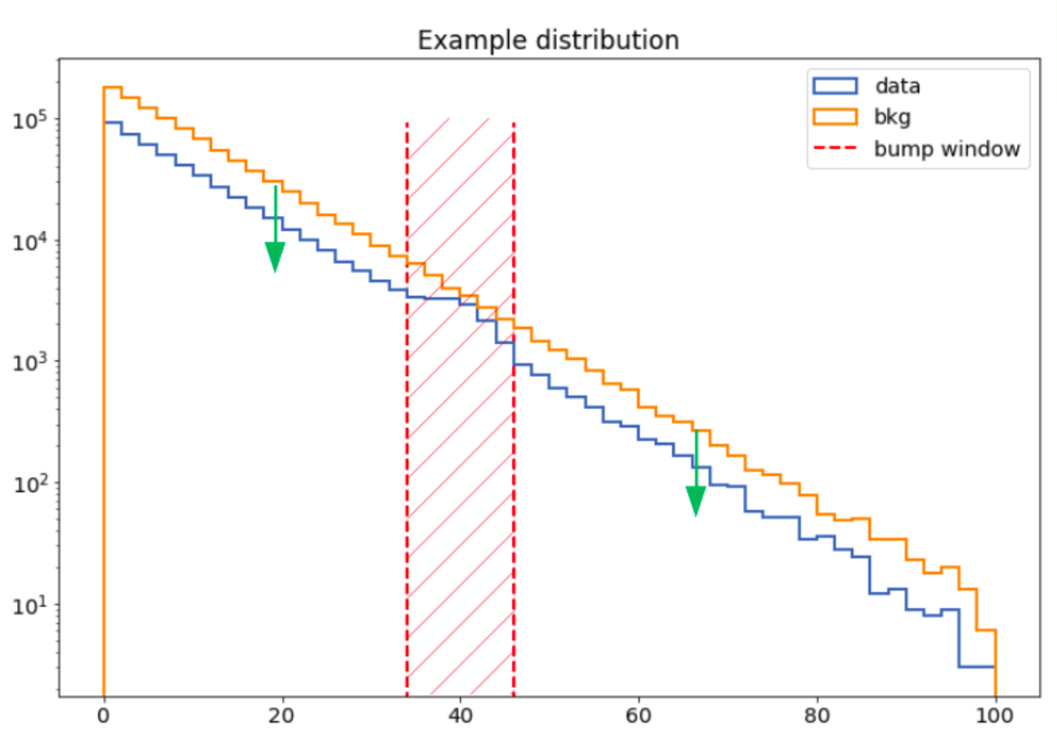
- Principle

  Automated normalization procedure

  Scale factor is computed for every tested interval

  Rescale the background to data

  No prior knowledge on the normalization



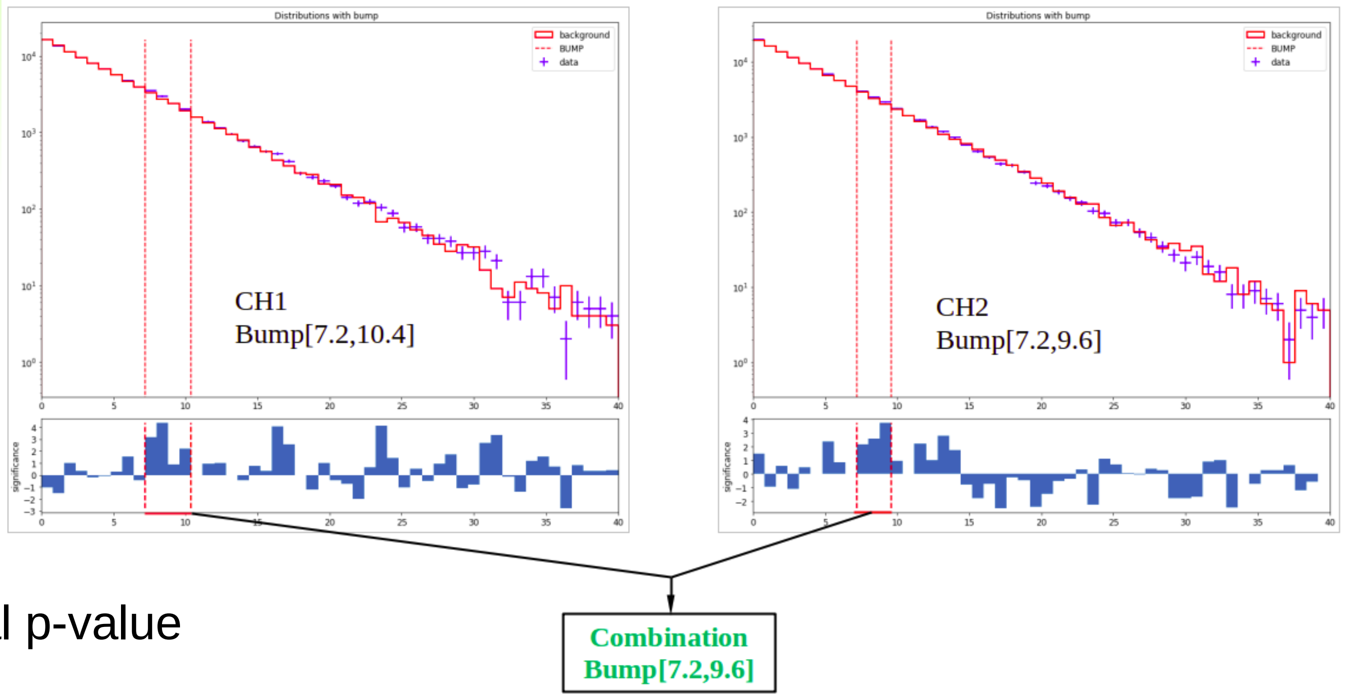
Enbaled with the ‘use_sideband’ setting

Multi-channel combination

- Principle

Look for a deviation in every channels

Combined bump
=
Intersection of individual bumps



Combined local p-value
=
product of individual local p-values

Coming in next release

Different combination techniques are under study

Multi-channel combination

- Alternative combnation technique

Fisher method (arXiv:1707.06897)

$$t_{comb} = -2\sum_{i=1}^{N} \ln(p_i) \equiv \chi^2_{2N}$$

N = number of channels
p_i = individual channel p-value

t_{comb} is distributed like a chi2 with 2N dof

=> $p_{comb} = 1 - \mathrm{cdf}_{\chi^2_{2N}}(t_{comb})$

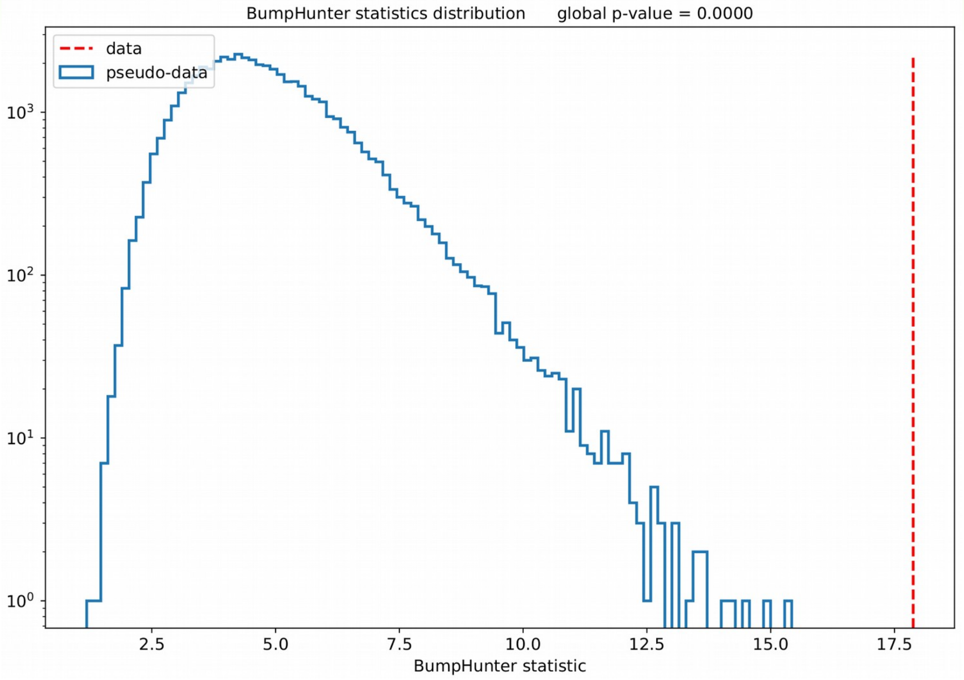
Two possibilities :

– Apply on local p-value and keep the overlap condition **(problem !)**

– Apply on global p-value and <u>no overlap condition</u>

Minimum p-value and fit

- Why a fit ?



In this example

global p-value = 0

=> σ = **+∞ !!**

Higher significance => need a **lot** of stat

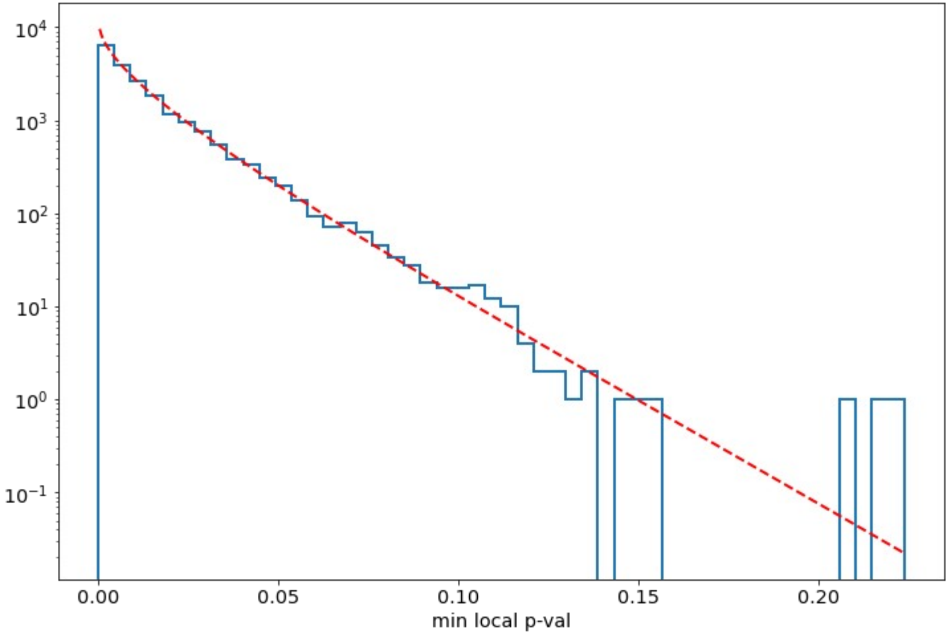
What to do ?

Fit the blue histogram

Minimum p-value and fit

- First attempt (***very*** *preliminary*)

p-value hacking (based on arXiv:1603.07532)



Fit of the **local p-value distribution**

$$\varphi_m(p;p_M) = m\, e^{erfc^{-1}(2p_M)\left(2erfc^{-1}(2p) - erfc^{-1}(2p_M)\right)} \left(1 - \frac{1}{2} erfc\left(erfc^{-1}(2p) - erfc^{-1}(2p_M)\right)\right)^{m-1}$$

m = number of independent test

p_M = True median of the distribution

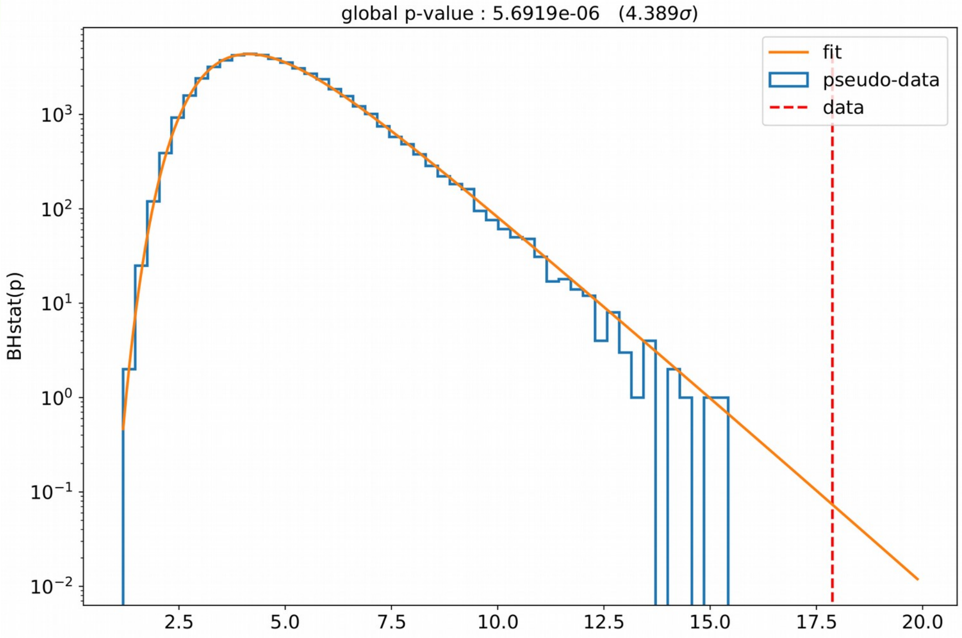
Reminder :

local p-value = min p-value among m tests

But, **tests are not independent**

Minimum p-value and fit

- Results (***very*** *preliminary*)



Transform the min p-value distribution into BumpHunter test statistic distribution (change of variable)

$$t = -\ln(p) \quad \Rightarrow \quad F(t) = \varphi[p(t)] \times \left| \frac{dp}{dt} \right|$$

It seems to fit !

Need more test to understand the behavior of the fit

Work in progress

Summary

- Public implementation of BumpHunter in python
- Integrated in Scikit-HEP
- Propose several extensions of the algorithm
- Next release is under reviewing
- Other nice new features are on the way

Thank you for your attention