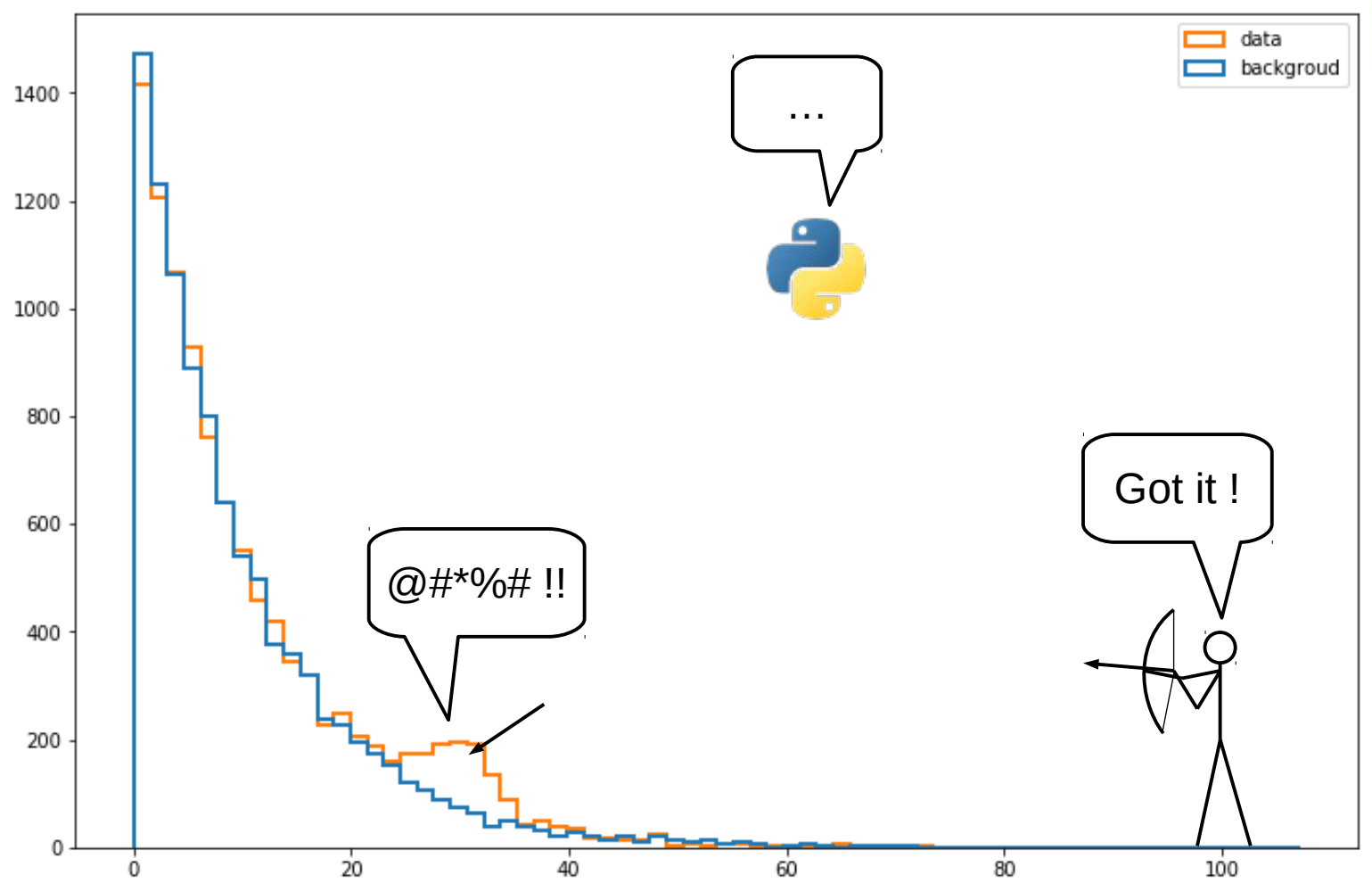
pyBumpHunter : a bump hunting tool in python



The BumpHunter algorithm

- Principle

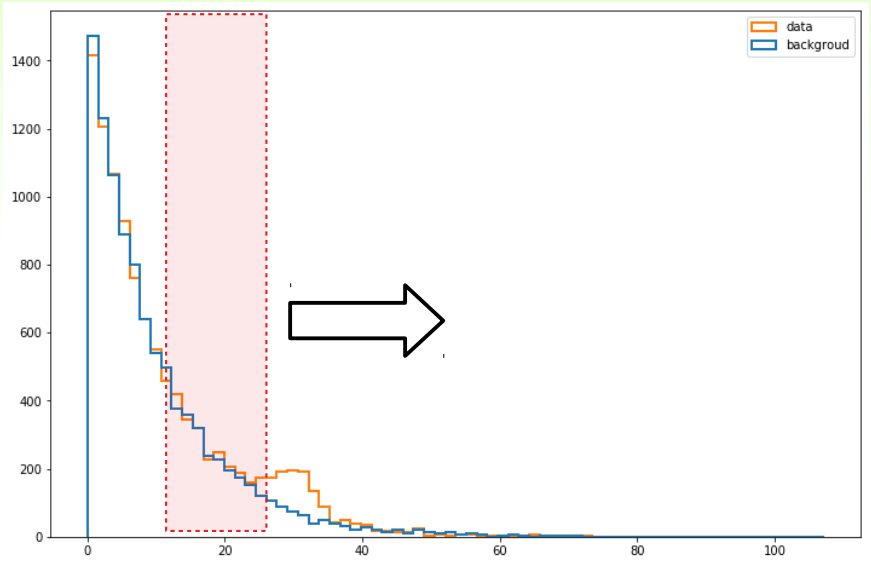
  Look for an **excess** (or deficit) in a data distribution w.r.t. a reference background

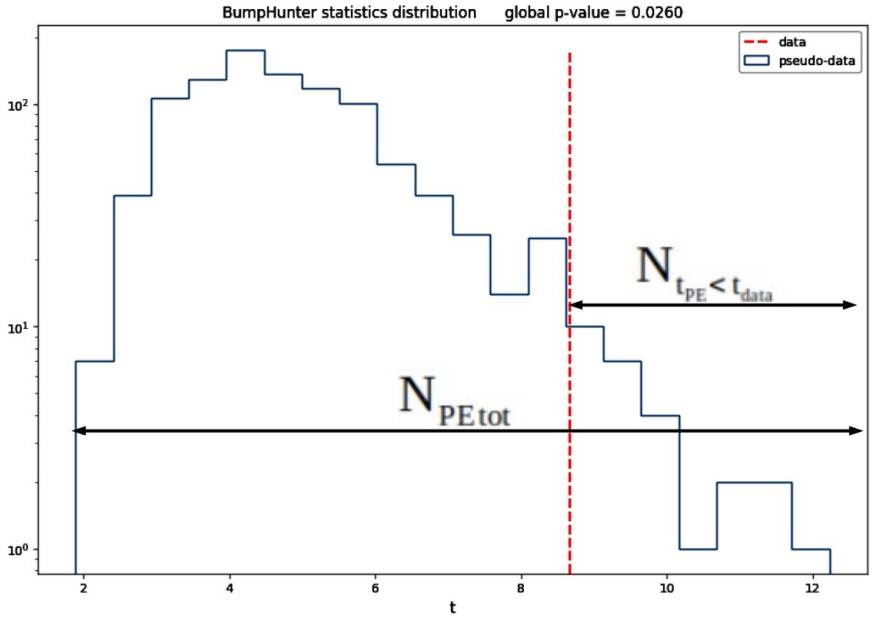
  Compute the local and global p-value of the localized deviation found in data

  The "*bump hunt*" is done using **histograms**

  BumpHunter generate **toys** from the reference and scans them like data (bkg-only local p-value distribution)

  The global p-value is the p-value of the local p-values





The BumpHunter algorithm

- Why do we need pyBumpHunter ?

  Model agnostic bump hunt
  Don’t need any signal model

  Possibility to use **data driven reference background**
  See Ioan’s talk …

  Pure python based (don’t need any ROOT setup to work)

  Code available on github

  The tool was presented at **PyHEP** last July (link)
  It has also been accepted in **Scikit-HEP**

  A paper presenting the tool and some tests of all the features is in preparation

Features proposed in pyBumpHunter

- Simple scan
- 2D scans

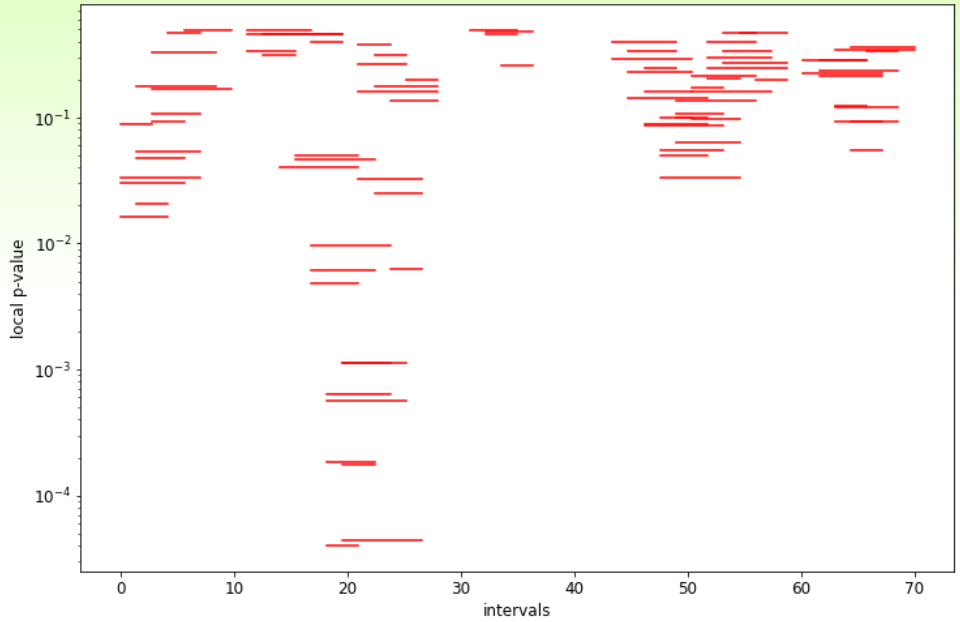
  Extension of the algorithm to 2D histograms
- Signal injection test (1D)

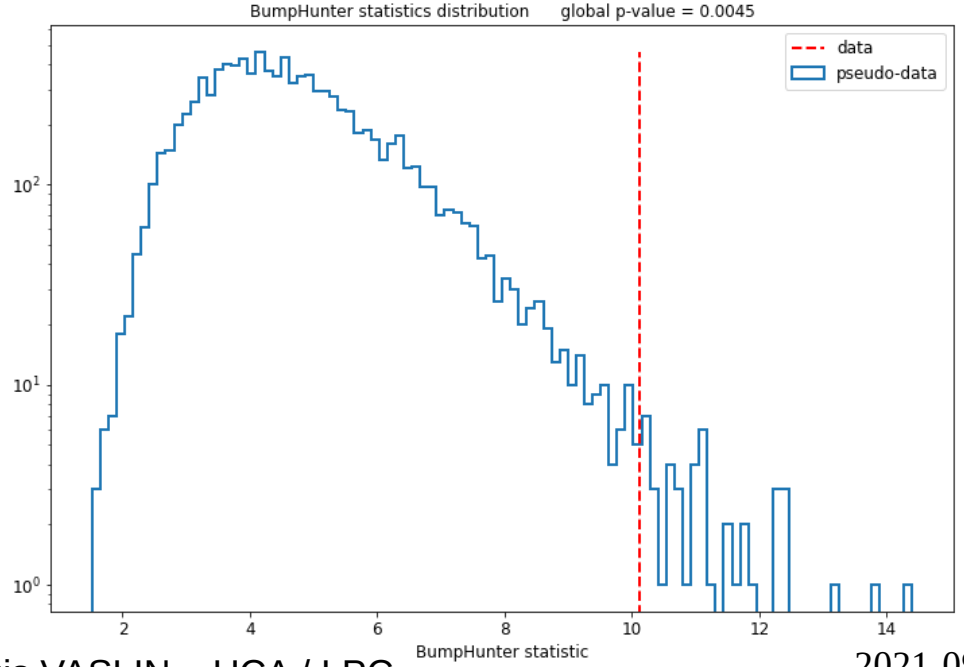
  To test sensitivity to a given signal model
- Side-band normalization

  To scale the reference background down to the data
- Multi-channel combination (*coming soon*)

  To get a combined local and global p-value over several channels

1D scan

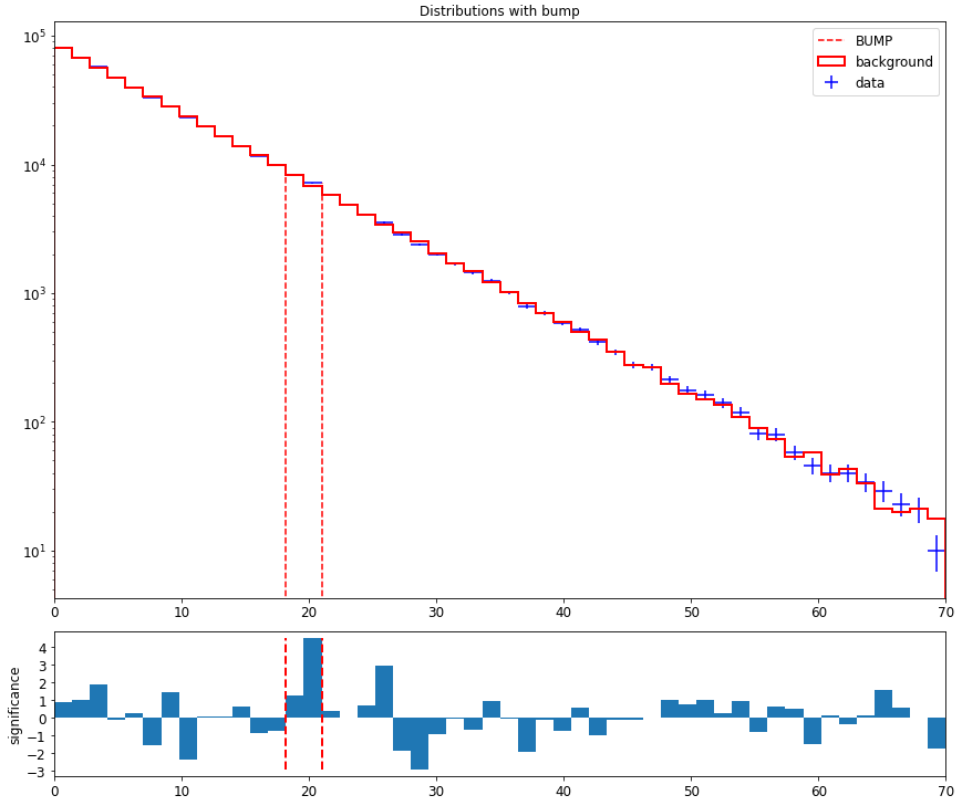




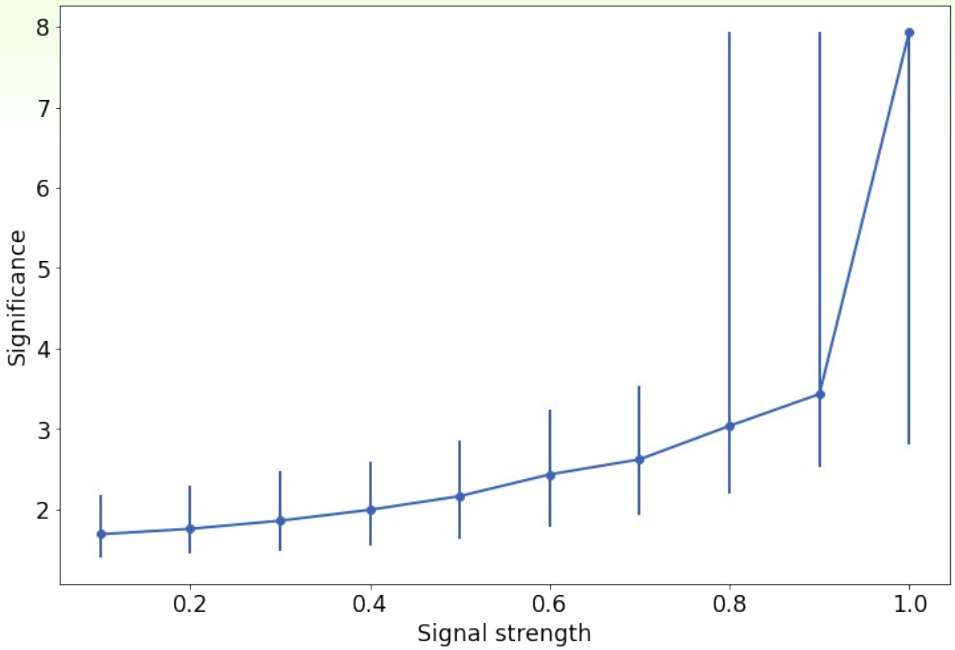
Follow the basic BumpHunter algorithm

Give the position and width of the bump, as well as the local and global p-value

Also give a *rough* evaluation of the signal content of the bump (data – ref)



Signal injection



Error bars obtained with bkg+signal toys
Points represent medians and error bars the 1st and 3rd quartiles

Saturation of the global significance when the local p-value becomes to small.
Producing more bkg-only toys can help for that

- Principle

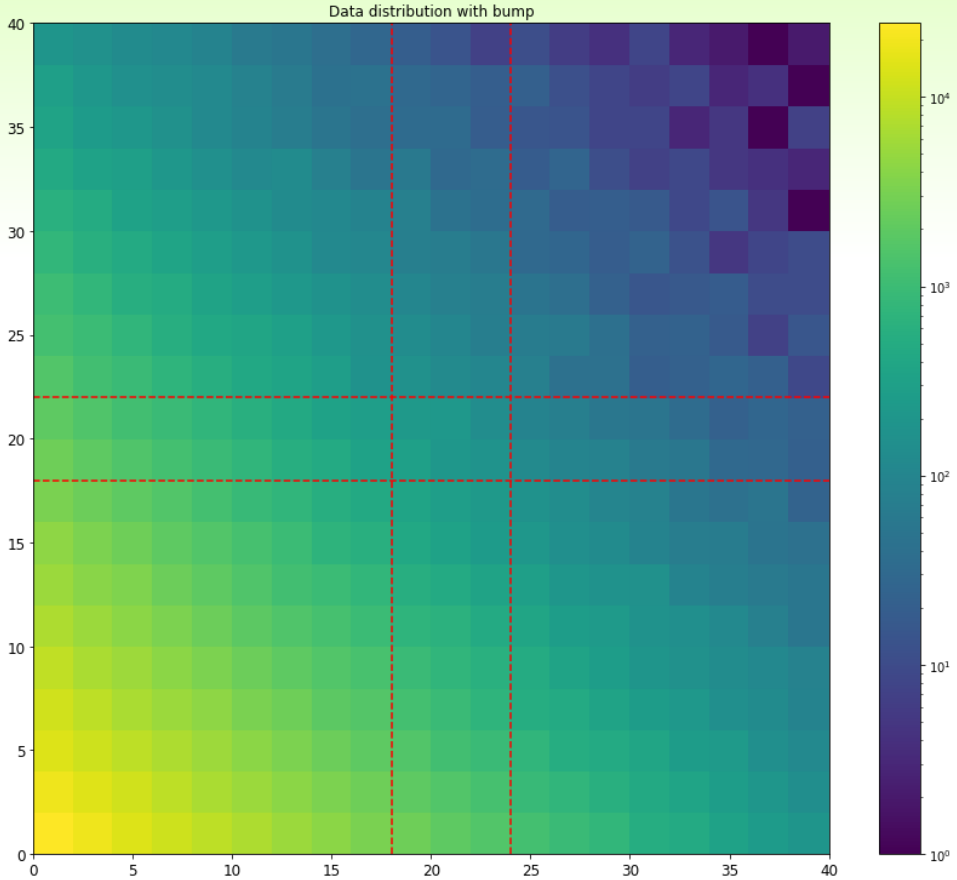
  Builds data based on a **reference background** and a **signal model**

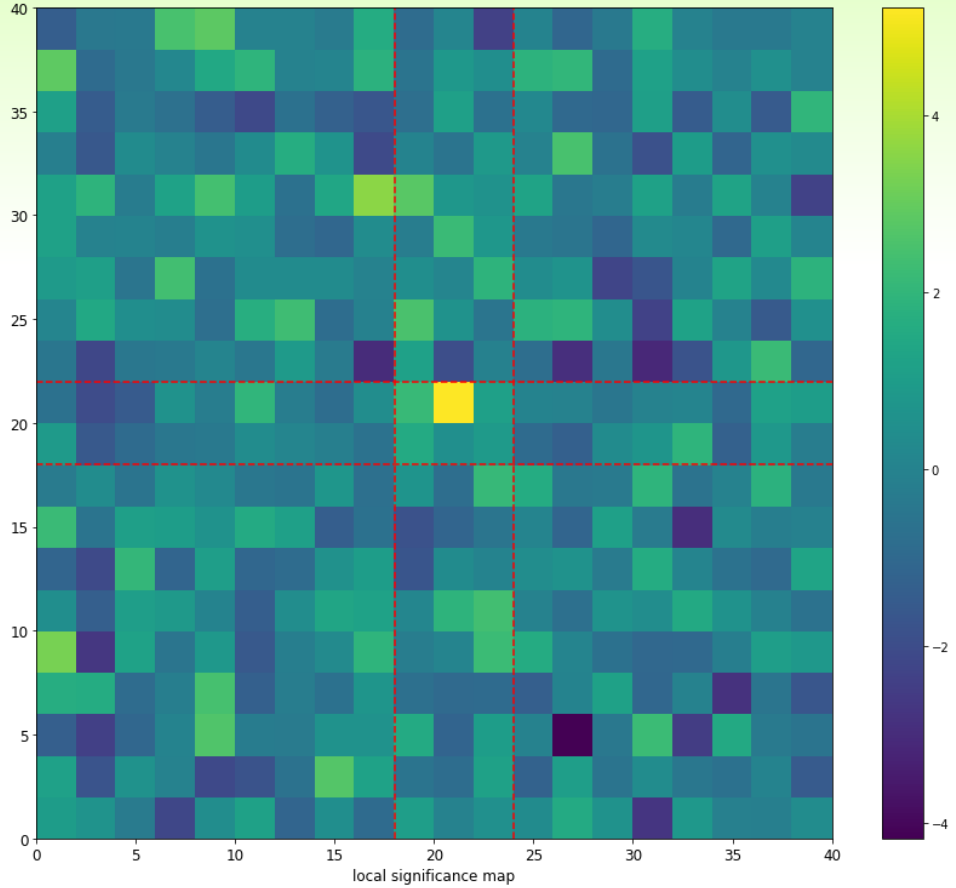
  Performs BumpHunter scans on the injected data with increasing signal strength

  Stops the process once the required global significance is reached

Signal strength is computed w.r.t. the expected number of signal event (model dependent)

2D BumpHunter



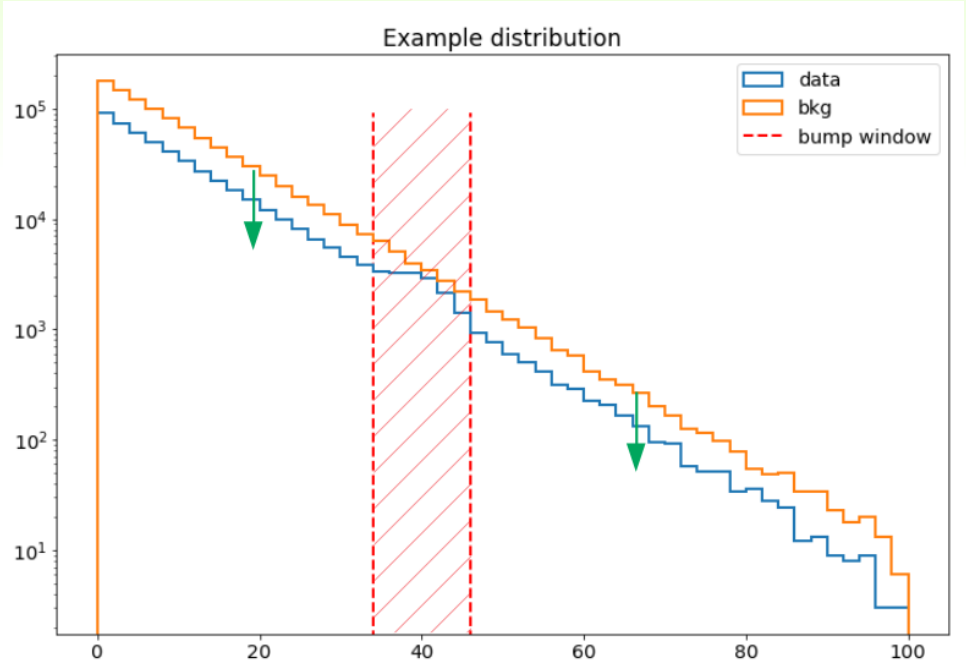


Works the same way as in 1D, but with 2D histograms

Plots show example with a Gaussian signal located at [20,20]
BumpHunter found it with ~3σ global p-value

Be careful with the statistics … in 2D the stat per bin is lower

Side-band normalization



- Principle

  For each tested interval, compute a **scale factor**

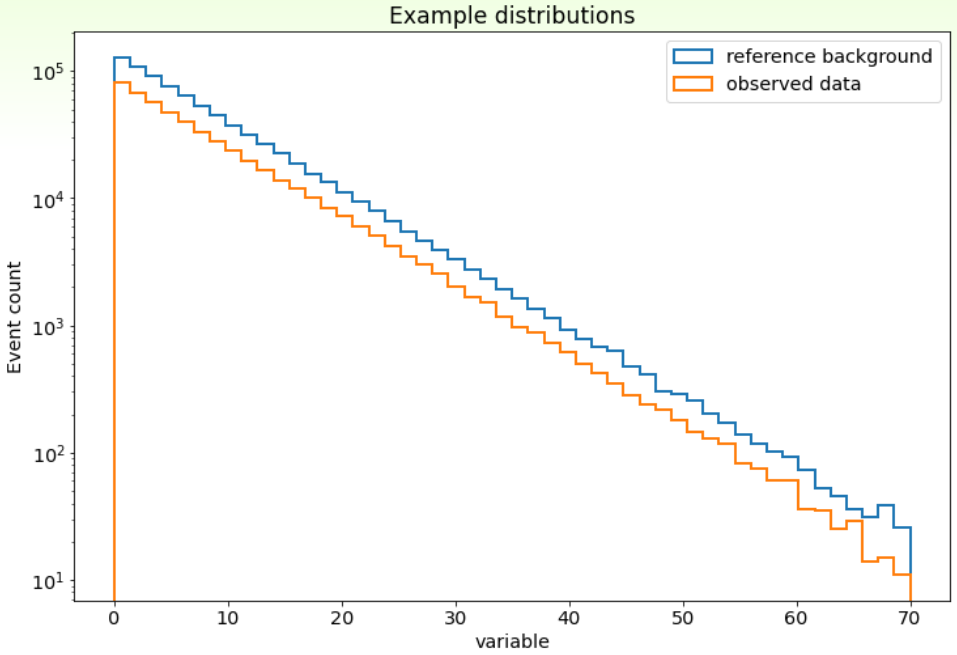
$$\text{scale} = \frac{\text{Ndata}_{\text{tot}} - \text{Ndata}_{\text{inter}}}{\text{Nref}_{\text{tot}} - \text{Nref}_{\text{inter}}}$$

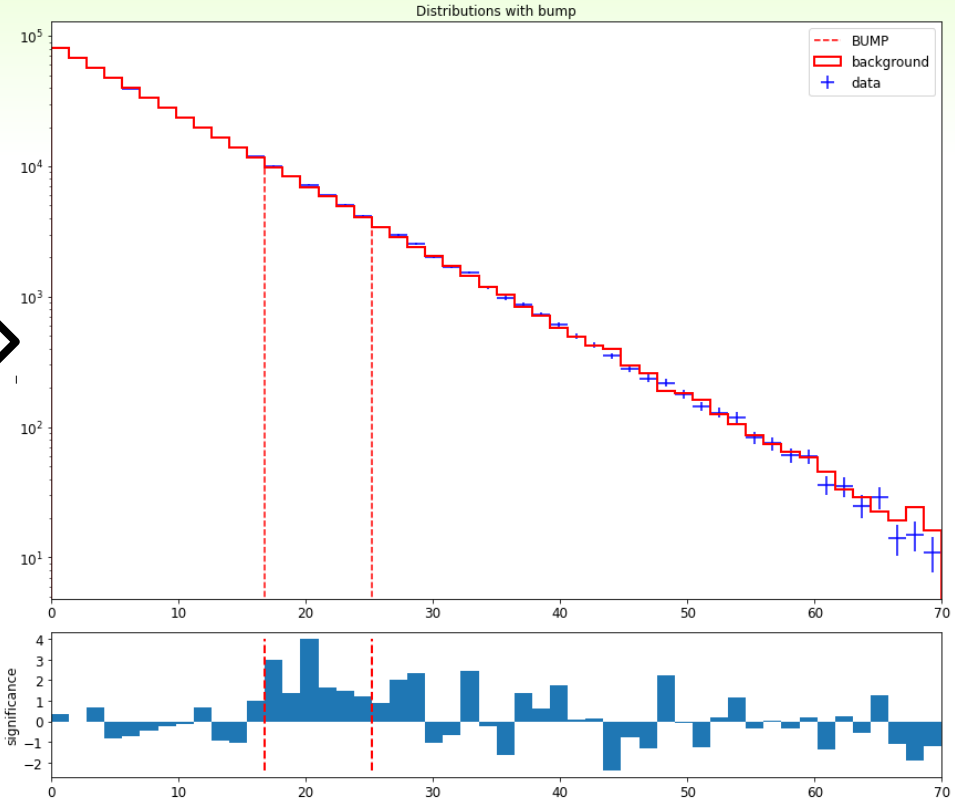
  Scale factor applied to the number of background event when computing local p-value

Objective :

Normalize the reference to data without any prior knowledge on expected background

Side-band normalization

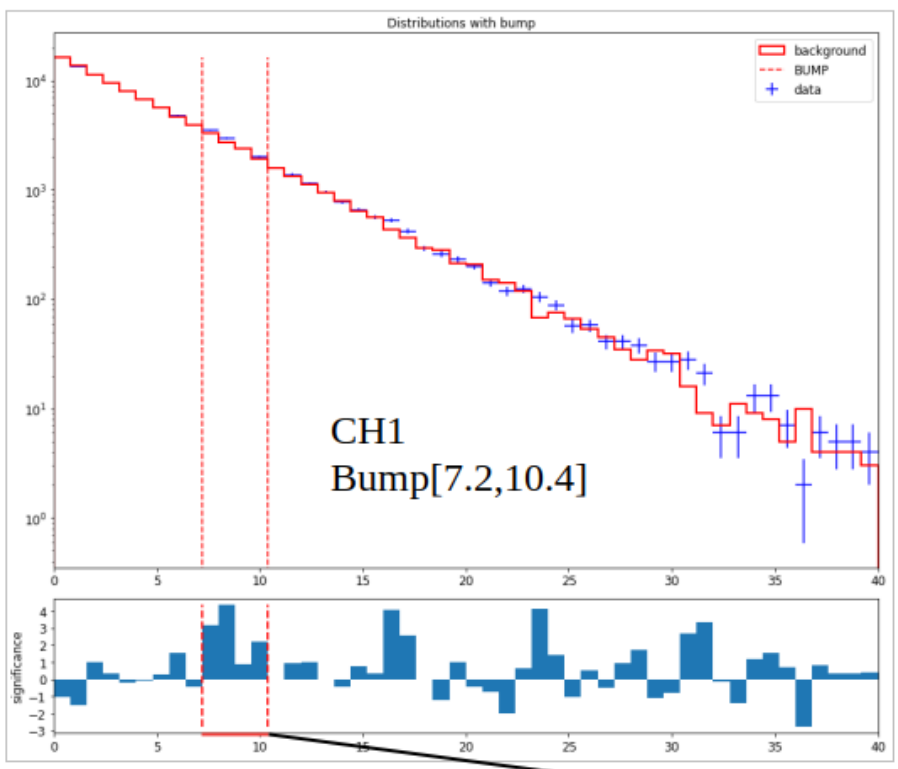


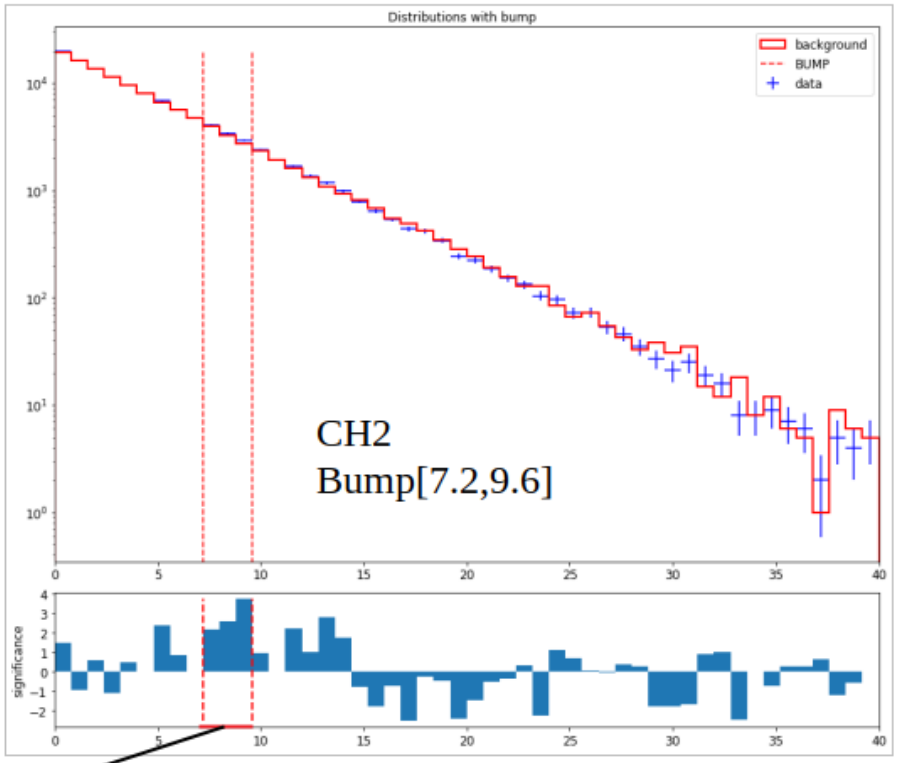


Reference : 800k generated event
Data : 500k background event + signal

Expected signal position (mean) : 20

Multi-channel combination





Combination
Bump[7.2,9.6]

The **combined bump window** is the intersection of each channel bump windows

Works with channels with **different binning**

combined local p-value
=
product of all individual p-values

This feature will be added in the **next release**

Conclusion

- pyBumpHunter is ready to use

  Pip installable and recently integrated to **Scikit-HEP**

  Many features are already available

  Presentation at PyHEP (with video recording available)

- Development is still ongoing

  New features and improvements will keep coming

  Multi-channel combination almost ready for the next release

  New features planed :
  - API refactoring (more efficient, both for developers and users)
  - Add treatment of systematic uncertainties
  - Anything else that could be useful (***your*** *ideas are welcome*)