Bracketing of CTA IRFs

Ievgen Vovk

CTA systematics call
28.05.2018

The case

CTA systematics plays an important role in establishing the performance when weak signals are looked for.

Best solution: evaluate the low-level instrument systematics distribution and propagate it to the distribution in IRFs.

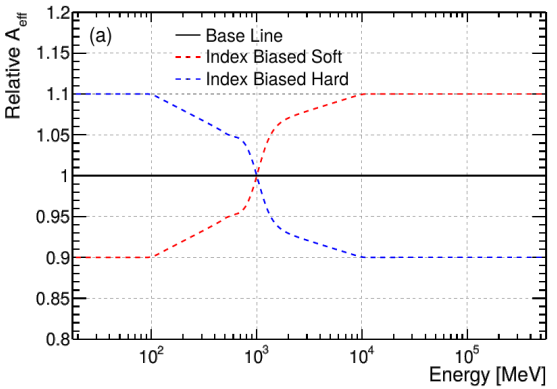
Threshold solution: evaluate the limits of low-level instrument systematics and get bracketed IRF.

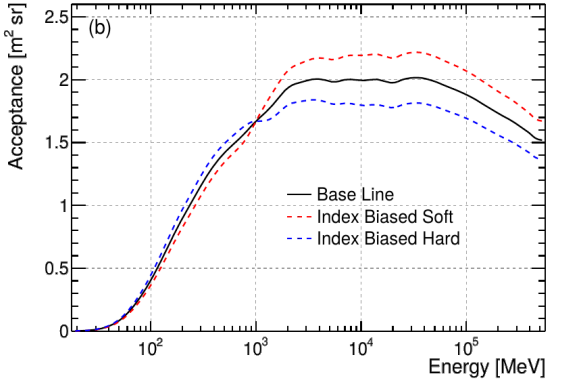
Poor-mans solution (short time scale plan): bracketing the IRFs using simply analytical scaling profiles and evaluate the performance in the limiting cases.

Fermi/LAT way:

$$A'_{eff} = A_{eff}(\theta, E)\,(1+\varepsilon(E)B(E))$$

$$B(E) = \tanh(1/k * \log(E/E_0))$$





Suggested bracketing of IRFs

1. PSF

$$\sigma_{1..n} \rightarrow \alpha\, \sigma_{1..n}$$

A simple scaling of the (sum of) gaussian PSF components

2. Collection area

$$A_{eff} \rightarrow A_{eff}(\theta, E) \times [1+\varepsilon_1(E)B_1(E)] \times [1+\varepsilon_2(\theta)B_2(\theta)]$$

$$B_1(E) = \tanh(1/k_1 \times \log(E/E_0)) \qquad B_2(\theta) = \tanh(1/k_2 \times \log(\theta/\theta_0))$$

This modifies both the energy and spatial response.

3. Saving / using

The resulting IRF is stored to the global data base, caldb index is updated (the original is backed up).

After this the IRF should be accessible by all CTA analysis routines – CTools, GammaPy etc.

Steps 1-2 can be implemented in Ctools/GammaPy, in which case the PDF of systematics should be also specified.

Pros: systematics included to the returned error bars, less CPU

Cons: Sensitive to sys distribution, more free params.

Alternatively, IRF Fits files can be updated with the limiting systematics cases.

Pros: No need to change existing software

Cons: Need to re-run every fit 2-3 times

Implementation: CalDB structure

"CALDB" variable holds the root path to the database location.
There may be several data bases inside:

```
user@machine:~$ tree -L 2 $CALDB/data/cta/
├── 1dc
│   ├── bcf
│   │   ├── North_z20_50h
│   │   ├── North_z40_50h
│   │   ├── South_z20_50h
│   │   └── South_z40_50h
│   └── caldb.indx
├── prod3b
│   ├── bcf
...
```

Updated IRFs can be stored in the corresponding "bcf" subfolders, without over-writing the existing files.

"caldb.indx" holds "name to path" information on IRFs and has to be appended after each update. It's a good idea to make a back up of it before starting any bracketing.

Automatic naming scheme (length-limited)

P-{psf_scale}_A-{energy_scale}_{energy_norm}_{energy_width}_{theta_scale}_{theta_norm}_{theta_width}

Ugly, but can be overridden by the user.

Implementation: for your code

A standalone class “CalDB” in Python:

```
caldb = CalDB(caldb_name, irf_name, verbose=False)
caldb.scale_irf(psf_scale,
                aeff_energy_scale, aeff_energy_norm, aeff_energy_transition_width,
                aeff_theta_scale, aeff_theta_norm, aeff_theta_transition_width,
                output_irf_file_name)
```

Each method has a docstring:

```
In [1]: from Scale_IRFs import CalDB
In [2]: CalDB.scale_irf?
Signature: Scale_IRFs.CalDB.scale_irf(self, psf_scale=1.0, aeff_energy_scale=0.0, aeff_energy_norm=1.0,
aeff_energy_transition_width=1.0, aeff_theta_scale=0.0, aeff_theta_norm=1.0, aeff_theta_transition_width=1.0,
output_irf_file_name='')
Docstring:
This method performs scaling of the loaded IRF - both PSF and Aeff, if necessary.
For the collection area two scalings can be applied: (1) vs energy and
(2) vs off-axis angle. In both cases the scaling function is taken as
(1 + scale * tanh((x-x0)/dx)). In case (1) the scaling value x is log10(energy).

Parameters
----------
psf_scale: float, optional
    The PSF scale factor. Each PSF sigma (there can be several!) will be multiplied by it.
    Defaults to 1.0 - equivalent of no scaling.
aeff_energy_scale: float, optional
```

The IRFs are stored under the predefined names and can be immediately used in the code.

Implementation: for your scripts

Callable script:

```
python Scale_IRF.py --caldb=”prod3b” --irf=”North_z20_50h” --psf_scale=0.5 --aeff_energy_scale=0.05
```

Has a detailed help:

```
user@machine:IRF scaling$ python Scale_IRFs.py --help
usage: Scale_IRFs.py [-h] [--caldb CALDB] [--irf IRF] [--psf_scale PSF_SCALE]

...

This script performs the scaling of the CTA calibration data base IRF
following the given settings. It is intended for the studies of the
systematics effects in CTA data analysis. Please note, that the existing data
base will be updated with the newly scaled IRF, i.e. your 'caldb.indx' file
will be over-written. Though a back up copy is created each time the script is
run, consider creating a master back up manually just in case.

optional arguments:
  -h, --help            show this help message and exit
  --caldb CALDB         Calibration data base name, e.g. "1dc"
  --irf IRF             The IRF to scale, e.g. "North_z20_50h"
```

Has the same functionality as the CalDB class itself.

Where to get it + final thoughts

The script/class can be found at EBL/ALP/LIV/IGMF publication page.

There’s no energy bracketing yet – can we also put it to IRFs?

What scaling ranges ε should we consider?

Any recommendations for the “transition width” k?

Should we do a more sophisticated spatial scaling of IRFs? Which one?