Full unbinned analysis - tailored approach

Giacomo D’Amico, Tim Unbehaun, Julia Djuvsland

$$N_{pred} \equiv \int dE \int d\mathbf{r} \;\; \phi(E, \mathbf{r})$$

Observed flux

$$-2 \log \mathcal{L} = 2 \cdot N_{pred} - 2 \sum_i \phi(E_i, \mathbf{r}_i)$$

Is there a way to get this quantity for all events (i=1,…, N) without having to define a binning in reconstructed energy and direction and without having to perform an interpolation?

For the i-th event with observed energy E_i and reconstructed direction $\mathbf{r_i}$

Observed flux / Energy Dispersion / PSF / Collection Area / True flux

$$\phi(E_i, \mathbf{r}_i) = \int dE' \int d\mathbf{r}' \;\; \mathrm{D}(E_i|E', \mathbf{r}') \times \mathrm{P}(\mathbf{r}_i|E', \mathbf{r}') \times \mathrm{A}(E', \mathbf{r}') \times \phi'(E', \mathbf{r}')$$

$$\Big\downarrow \text{ Numerical integration}$$

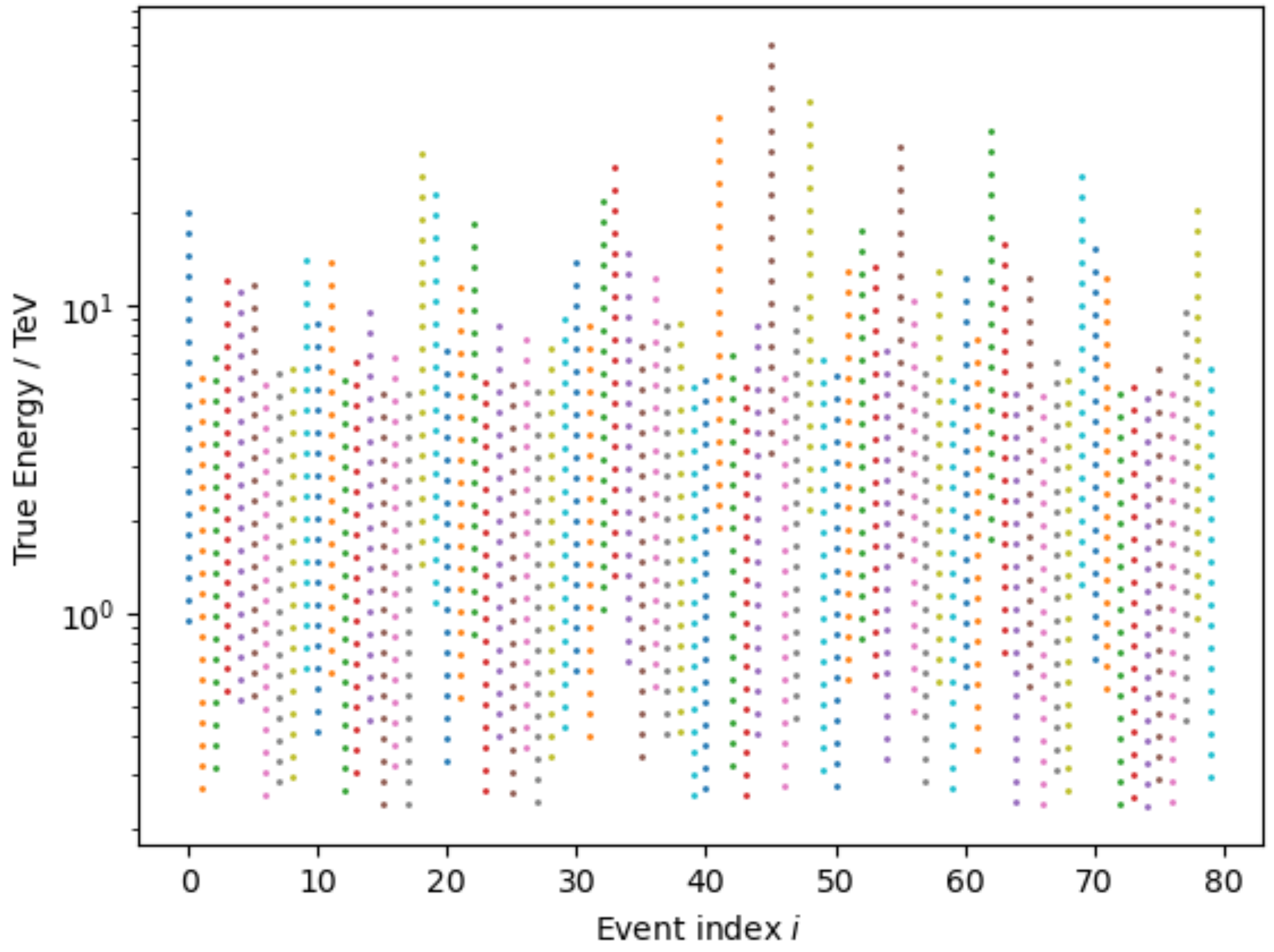
$$\phi(E_i, \mathbf{r}_i) = \sum_k \Delta E'_k \; \sum_{l,b} \Delta \mathbf{r}'_{l,b} \;\; \mathrm{D}(E_i|E'_k, \mathbf{r}'_{l,b}) \times \mathrm{P}(\mathbf{r}_i|E'_k, \mathbf{r}'_{l,b}) \times \mathrm{A}(E'_k, \mathbf{r}'_{l,b}) \times \phi'(E'_k, \mathbf{r}'_{l,b})$$

i : index from 1 to **total number of events**
k : index of binning in **true energy**
l : index of binning in **right ascension**
b : index of binning in **declination**

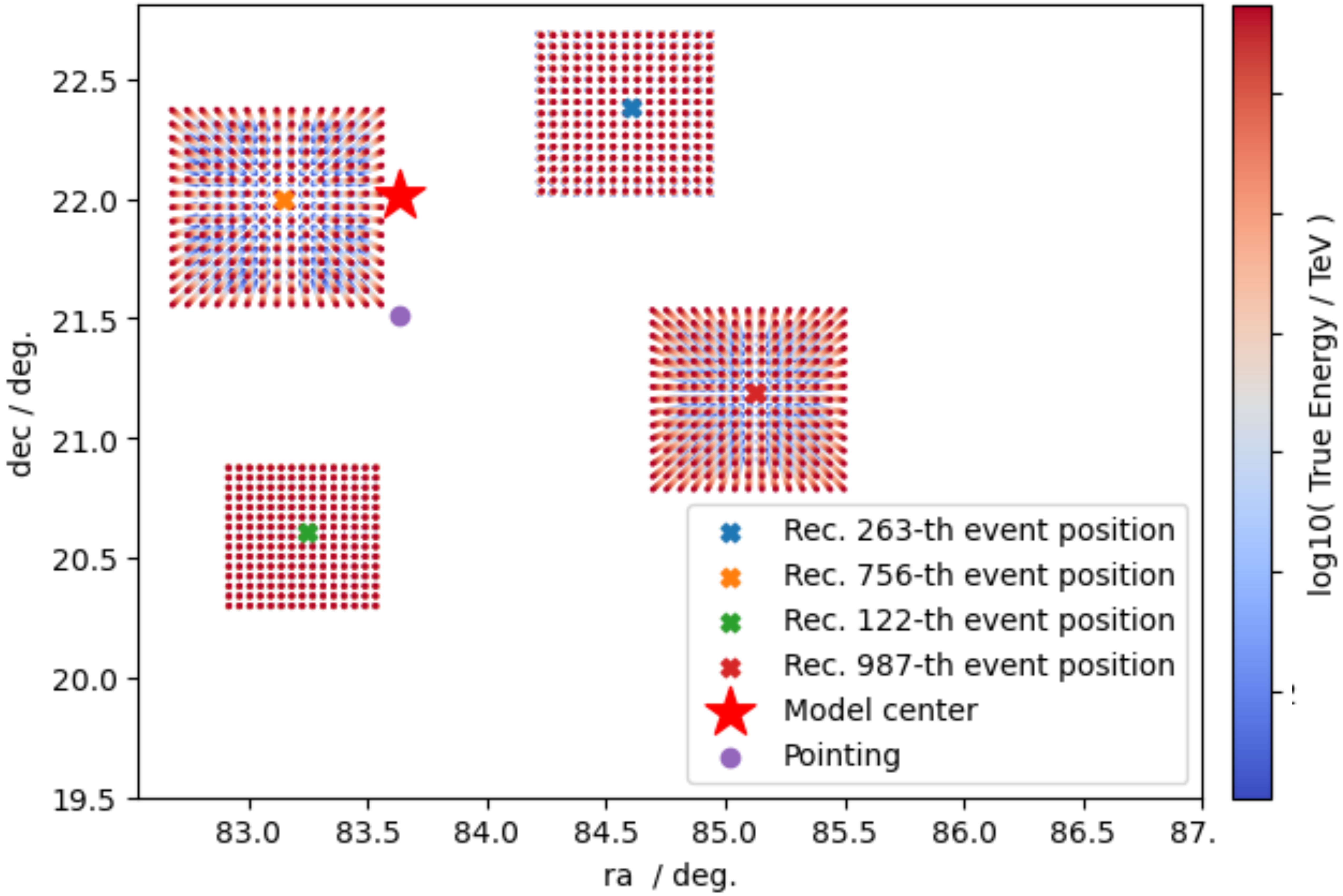
$$\Big\downarrow \text{ Event-Tailored binning}$$

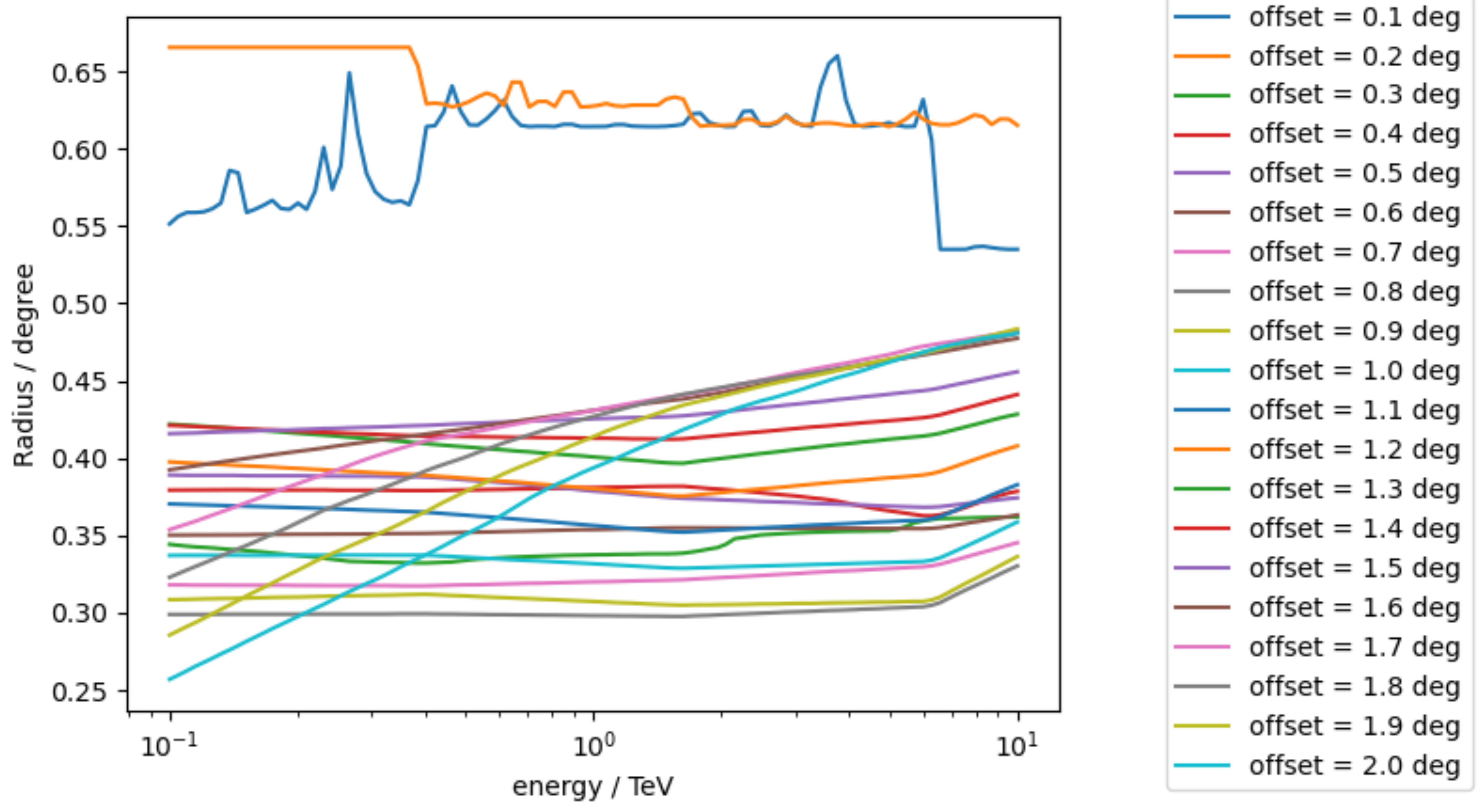
$$\phi(E_i, \mathbf{r}_i) = \sum_k \Delta E'_{i,k} \; \sum_{l,b} \Delta \mathbf{r}'_{i,k,l,b} \;\; \mathrm{D}(E_i|E'_{i,k}, \mathbf{r}'_{i,k,l,b}) \times \mathrm{P}(\mathbf{r}_i|E'_{i,k}, \mathbf{r}'_{i,k,l,b}) \times \mathrm{A}(E'_{i,k}, \mathbf{r}'_{i,k,l,b}) \times \phi'(E'_{i,k}, \mathbf{r}'_{i,k,l,b})$$

$$E'_{i,k}$$ shape = (N_i , N_k)



$$\mathbf{r}'_{i,k,l,b}$$ shape = (N_i , N_k , N_l , N_b)



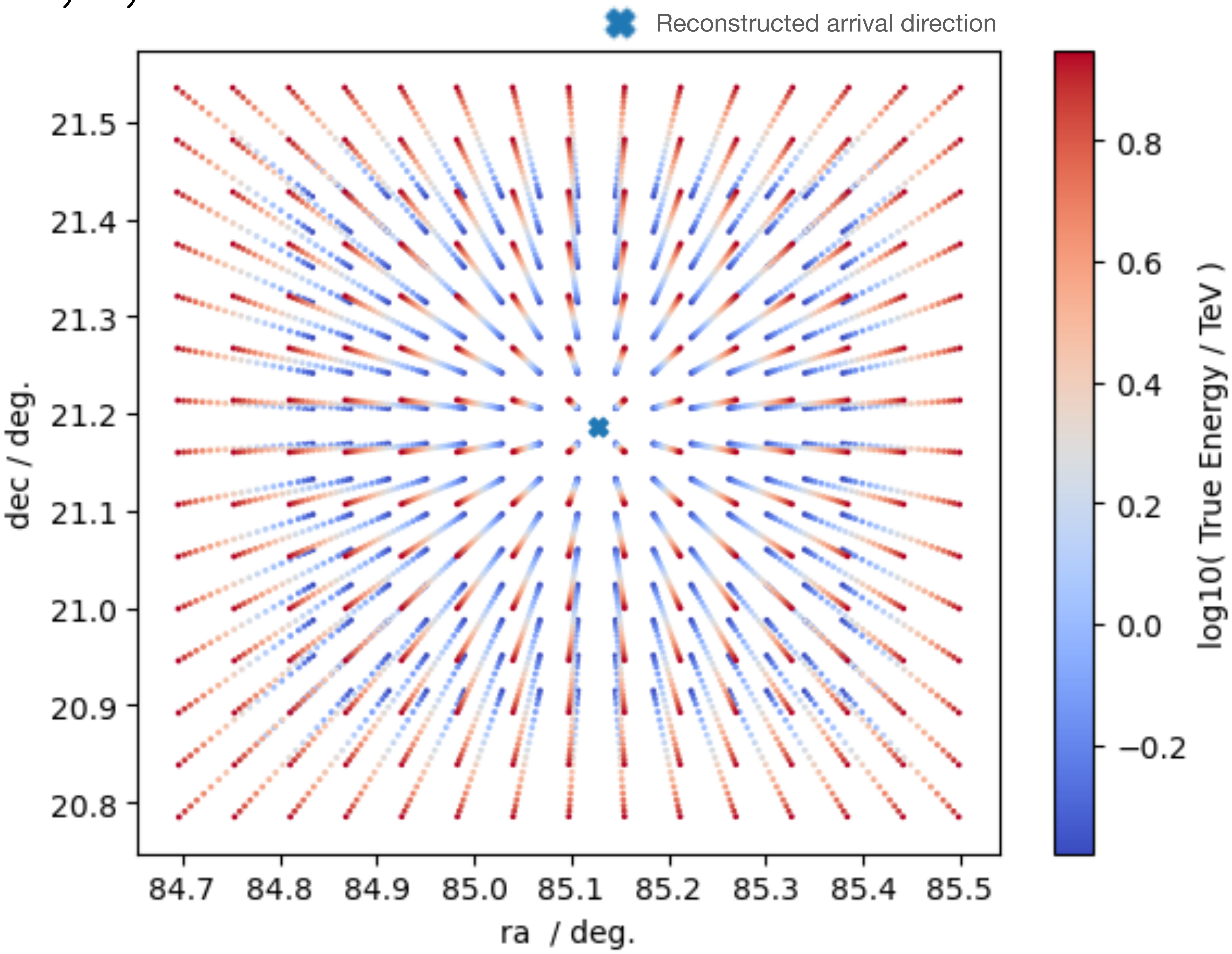


95% containment radius

$\mathbf{r}'_{i,k,l,b}$

i-th event for which

Obs. energy = 1.7 TeV
Obs. ra, dec = 85.1 deg , 21.2 deg



$$\phi(E_i, \mathbf{r}_i) = \sum_k \Delta E'_{i,k} \sum_{l,b} \Delta \mathbf{r}'_{i,k,l,b} \ \mathrm{D}(E_i | E'_{i,k}, \mathbf{r}'_{i,k,l,b}) \times \mathrm{P}(\mathbf{r}_i | E'_{i,k}, \mathbf{r}'_{i,k,l,b}) \times \mathrm{A}(E'_{i,k}, \mathbf{r}'_{i,k,l,b}) \times \phi'(E'_{i,k}, \mathbf{r}'_{i,k,l,b})$$

taking into account the circular symmetry of the PSF

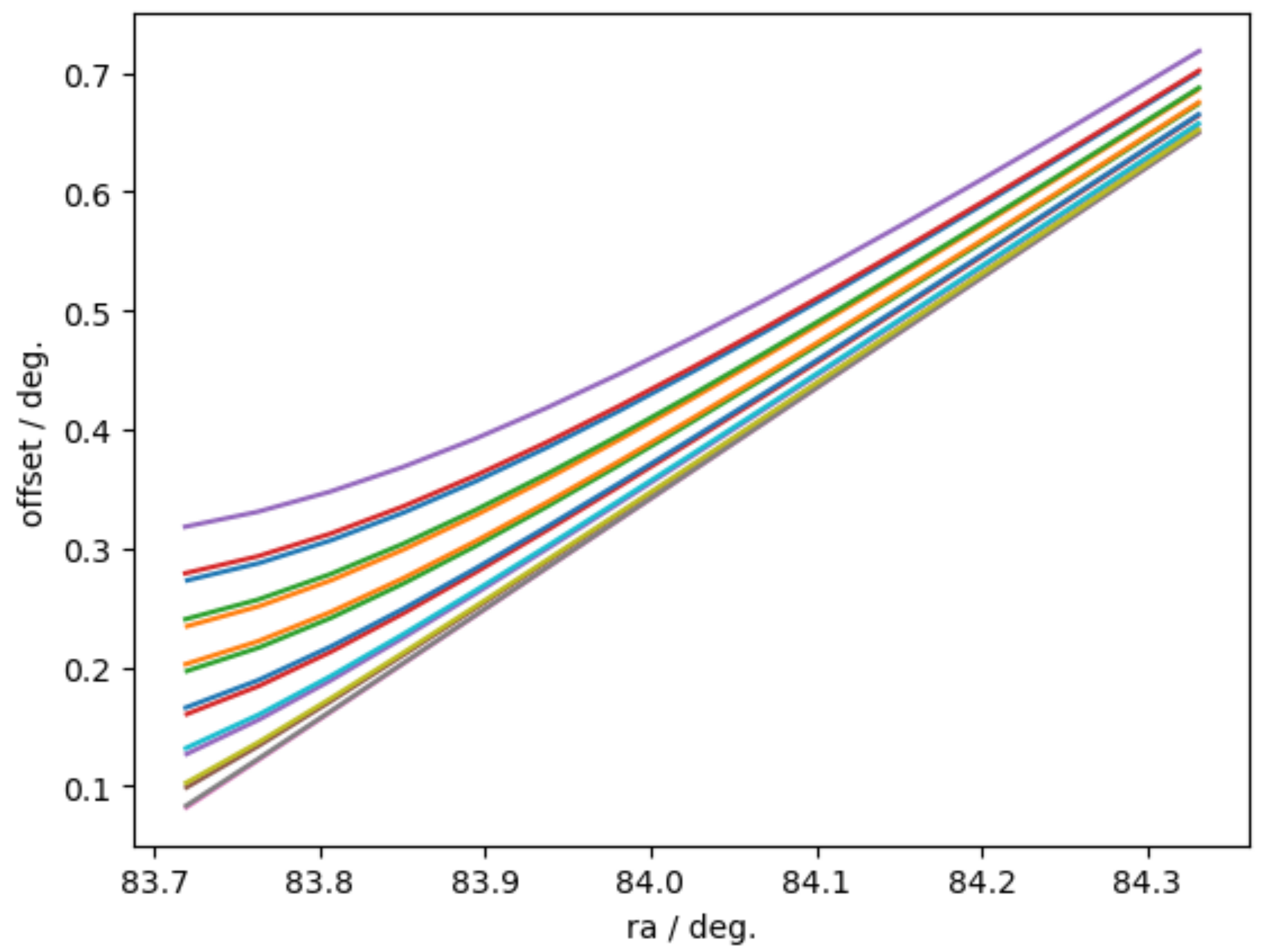
$$\phi(E_i, \mathbf{r}_i) = \sum_k \Delta E'_{i,k} \sum_{l,b} \Delta \mathbf{r}'_{i,k,l,b} \mathrm{D}(E_i | E'_{i,k}, \mathrm{O}_{i,k,l,b}) \times \mathrm{P}(d_{i,k,l,b} | E'_{i,k}, \mathrm{O}_{i,k,l,b}) \times \mathrm{A}(E'_{i,k}, \mathrm{O}_{i,k,l,b}) \times \phi'(E'_{i,k}, \mathbf{r}'_{i,k,l,b})$$

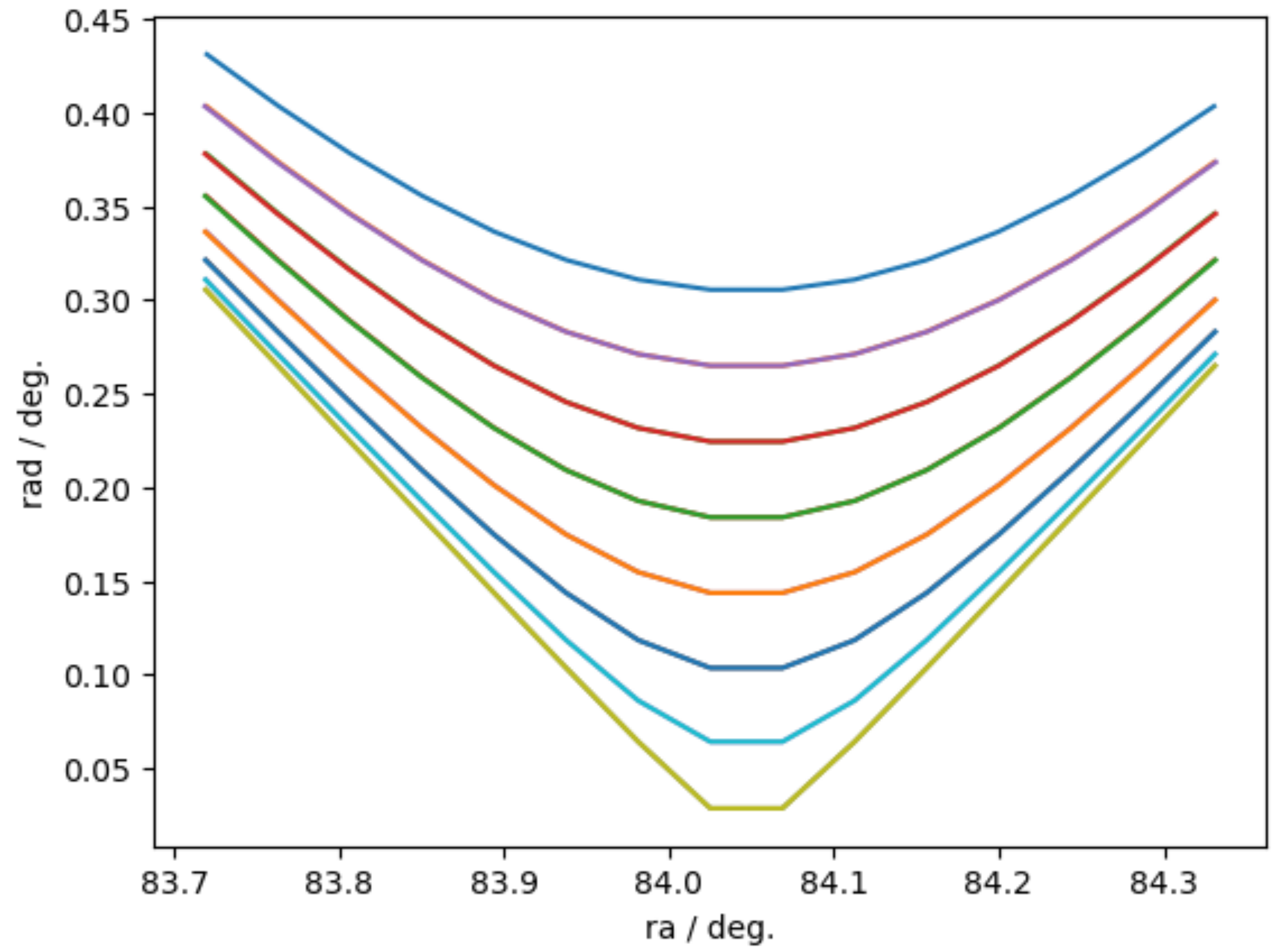
$$\mathrm{O}_{i,k,l,b} = |\mathbf{r}_i - \mathrm{pointing}|$$

"Offset" = distance between the telescope pointing and the reconstructed direction

$$d_{i,k,l,b} = |\mathbf{r}_i - \mathbf{r}'_{i,k,l,b}|$$

"rad" = distance between the true and the reconstructed direction



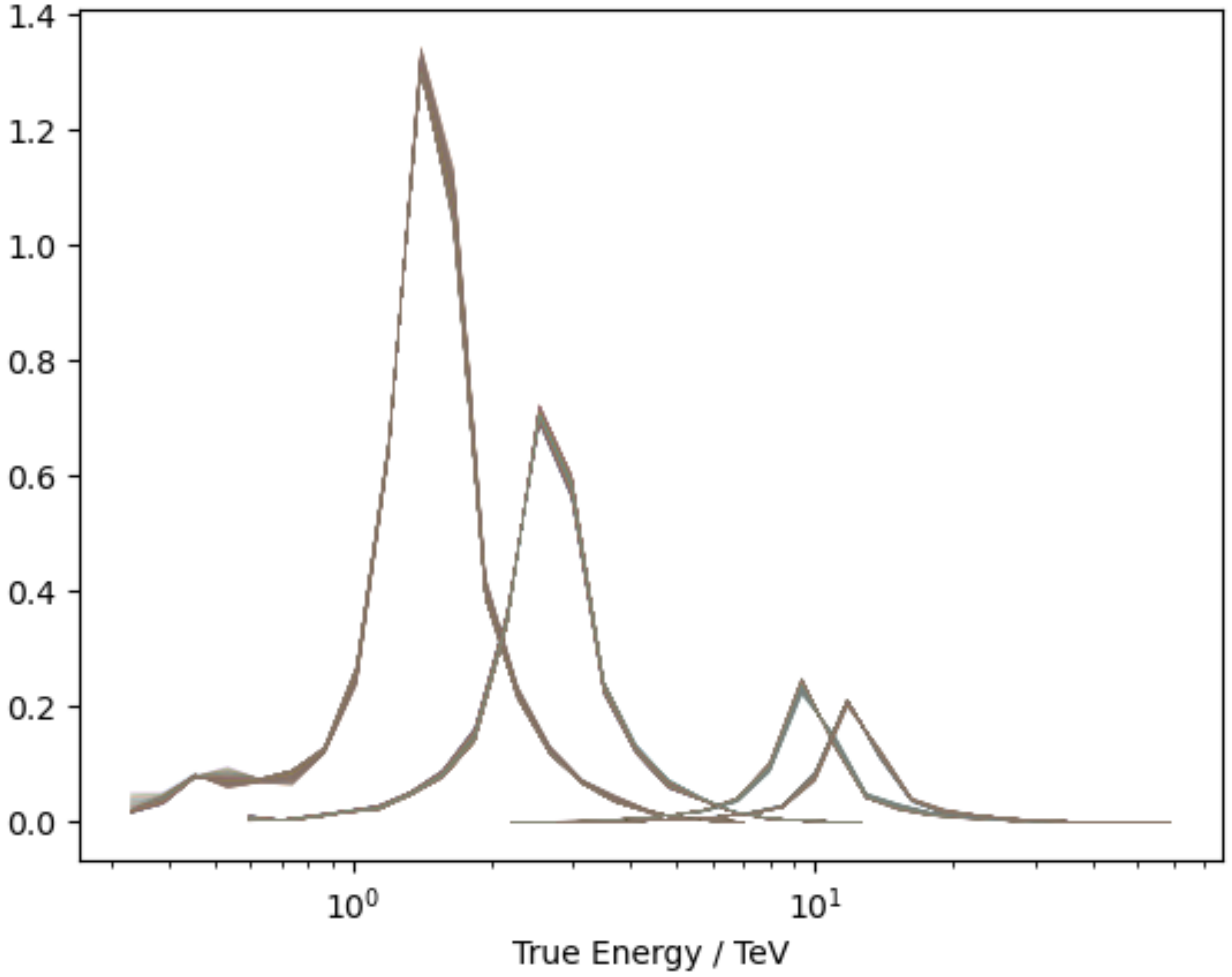


Energy Dispersion

Unit = 1 / TeV

$$D(E_i | E'_{i,k}, O_{i,k,l,b})$$

Shape = (N_i , N_k , N_l , N_b)



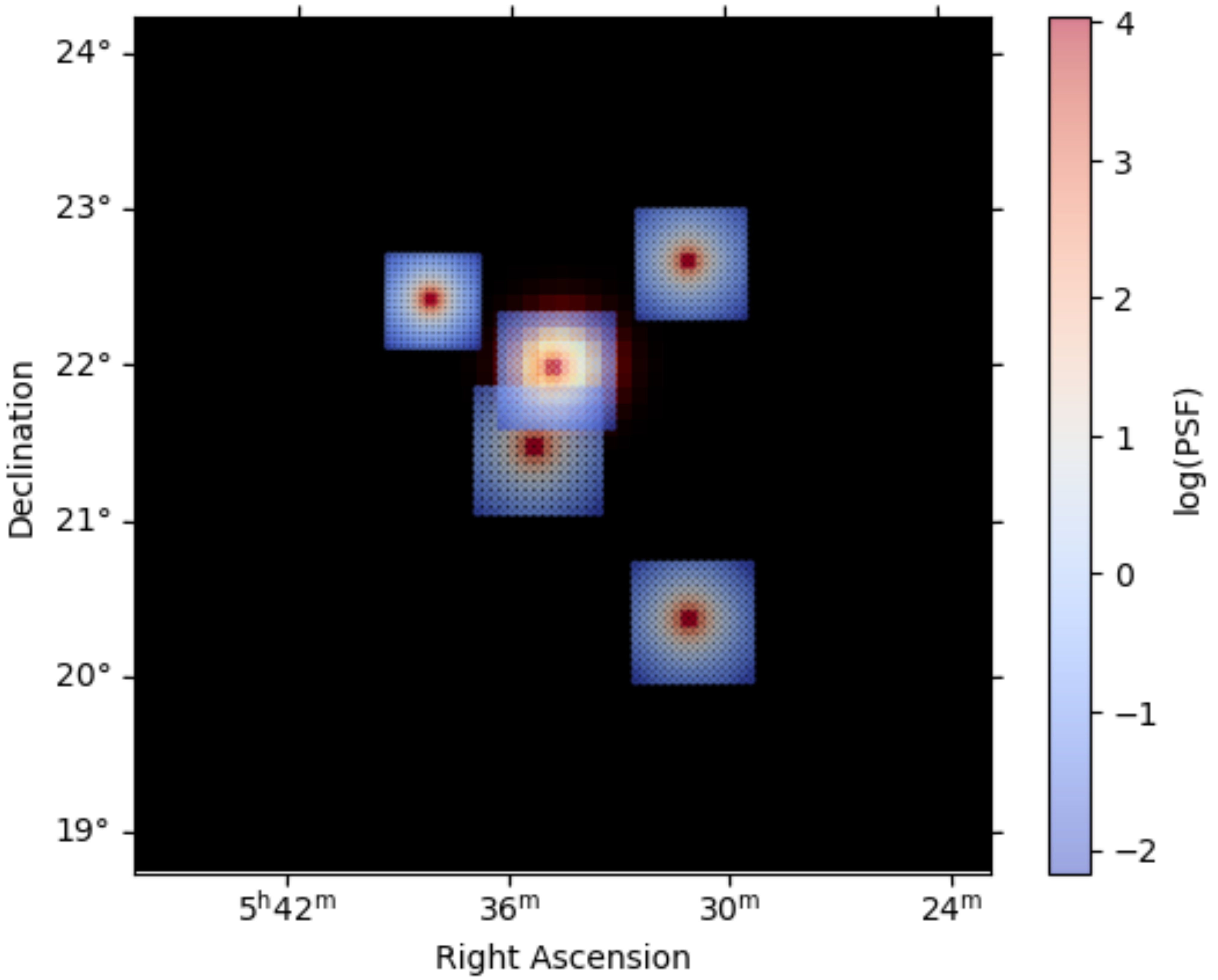

```
ereco_events       =  events.energy[:,None,None,None]

edisp2D        = observation.edisp
edisp2D.normalize()
edisp        =  edisp2D.evaluate(
                 offset =true_offset,
                 energy_true=etrue,
                 migra=ereco_events/etrue
               ) / etrue
```

PSF

Unit = 1 / deg^2

$$P(d_{i,k,l,b} | E'_{i,k}, O_{i,k,l,b})$$

Shape = (N_i , N_k , N_l , N_b)




```
psf3d         = observation.psf
psf3d.normalize()

psf_factors = psf3d.evaluate(
                 energy_true=etrue,
                 rad=rad,
                 offset=true_offset
             )
```

Seen as a tensorial product

Obs. Flux			Delta Energy		Delta Omega		Edisp		PSF		Coll. Area		True Flux
(N_i)	$=$	$\sum_{k,l,b}$	(N_i, N_k)	x	(N_i, N_l, N_b)	x	(N_i, N_k, N_l, N_b)	x	(N_i, N_k, N_l, N_b)	x	(N_i, N_k, N_l, N_b)	x	(N_i, N_k, N_l, N_b)
1 / (deg^2 x TeV)			TeV		deg^2		1 / TeV		1 / deg^2		s x m^2		1 / (s x m^2 x deg^2 x TeV)

Obs. Flux			IRF		True Flux
(N_i)	$=$	$\sum_{k,l,b}$	(N_i, N_k, N_l, N_b)	x	(N_i, N_k, N_l, N_b)
1 / (deg^2 x TeV)			m^2		1 / (s x m^2 x deg^2 x TeV)

Model independent
It has to be computed only **once** in the initialisation of the class

Model dependent
This is the only part that change during the fit and has to be computed many times

Example with:

$(N_i, N_k, N_l, N_b) = (1221, 20, 15, 15)$

Getting the full IRF takes around 5 seconds

Example with:

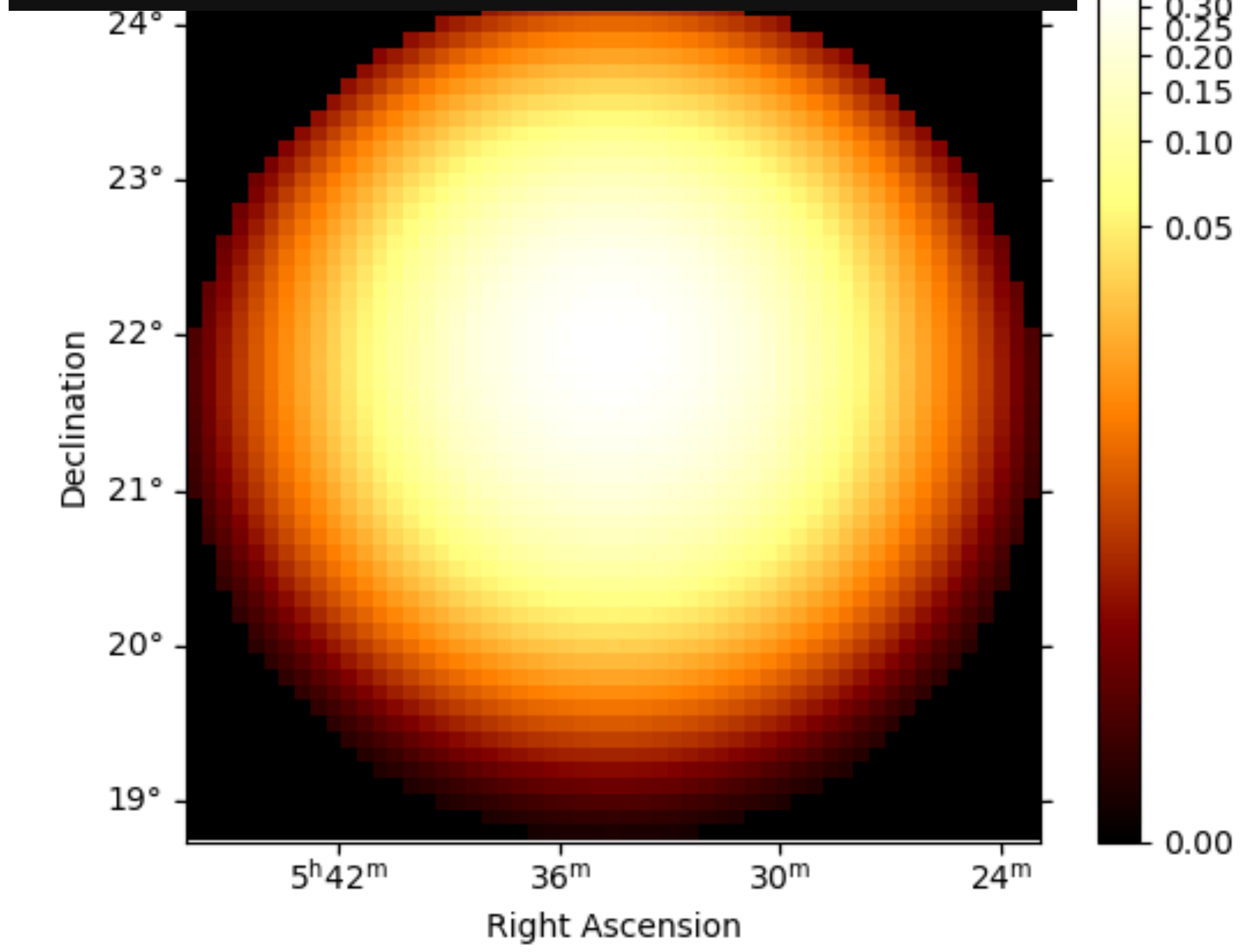
$(N_i, N_k, N_l, N_b) = (1221, 20, 15, 15)$

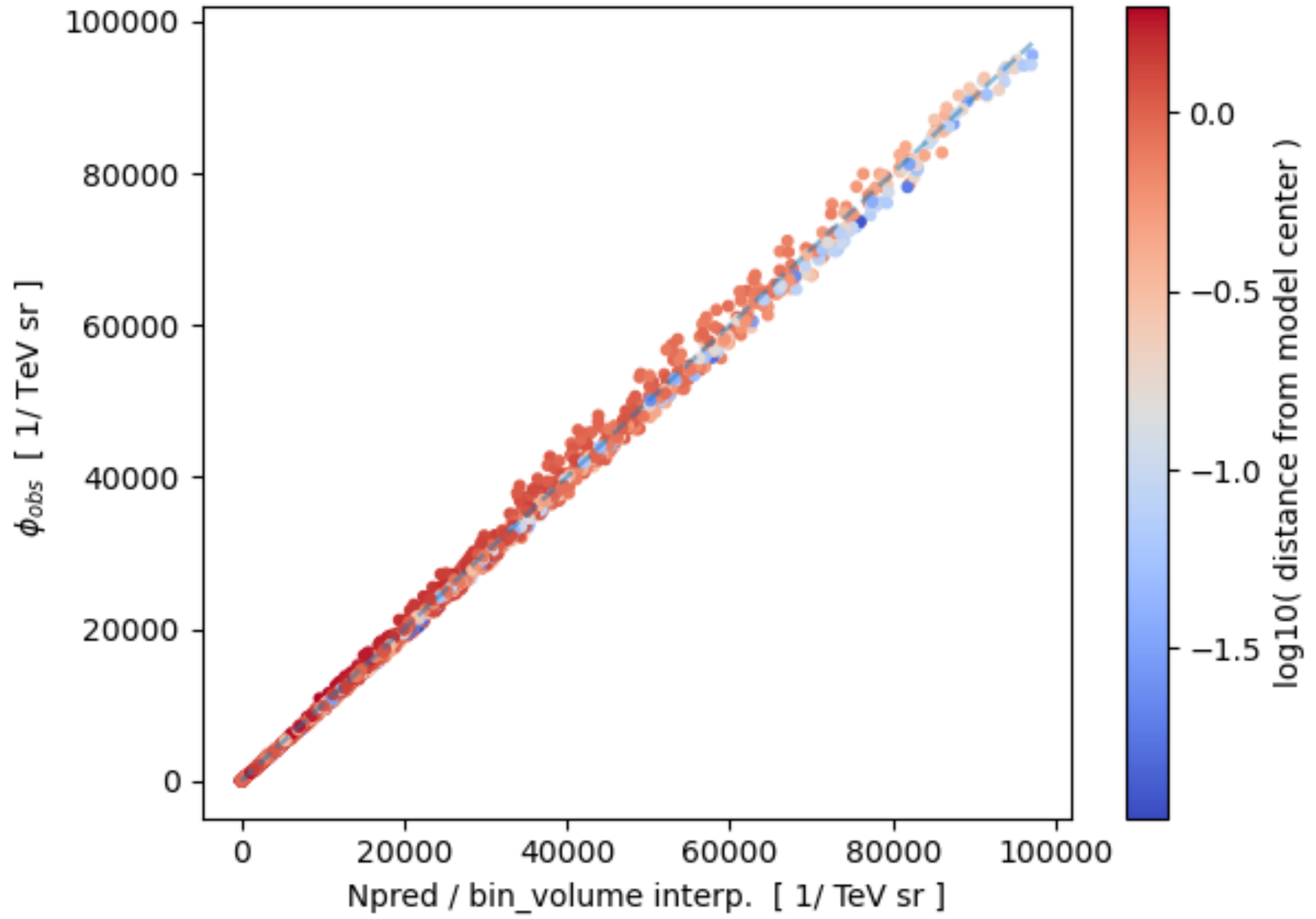
Computing the “True Flux” **tensor** and **convolving** it with IRF takes around 200 ms!

Sigma = 1 deg.

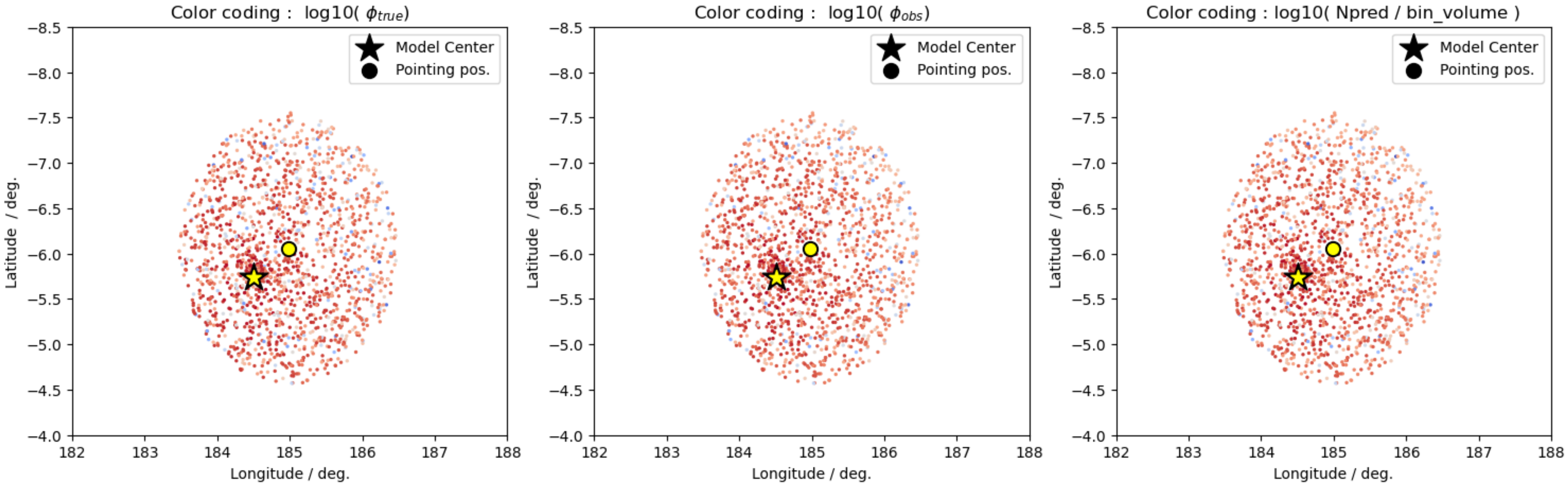
```
model_gauss = SkyModel(
    spatial_model  = GaussianSpatialModel(     lon_0      = "184.557 deg",            lat_0      = "-5.784 deg", sigma = '1 deg', frame = 'galactic'),
    spectral_model = LogParabolaSpectralModel( amplitude ='3.5e-11 cm-2 s-1 TeV-1', reference = '1 TeV',      alpha = 1.8,        beta  = 0.4),
    name           = 'crab_model_gauss'
)
```

```
mapnpred       = dataset.npred()
mapnpred       = dataset.evaluators['crab_model_gauss'].compute_npred()

mapnpred.sum_over_axes().plot(add_cbar=True,stretch='log')
```





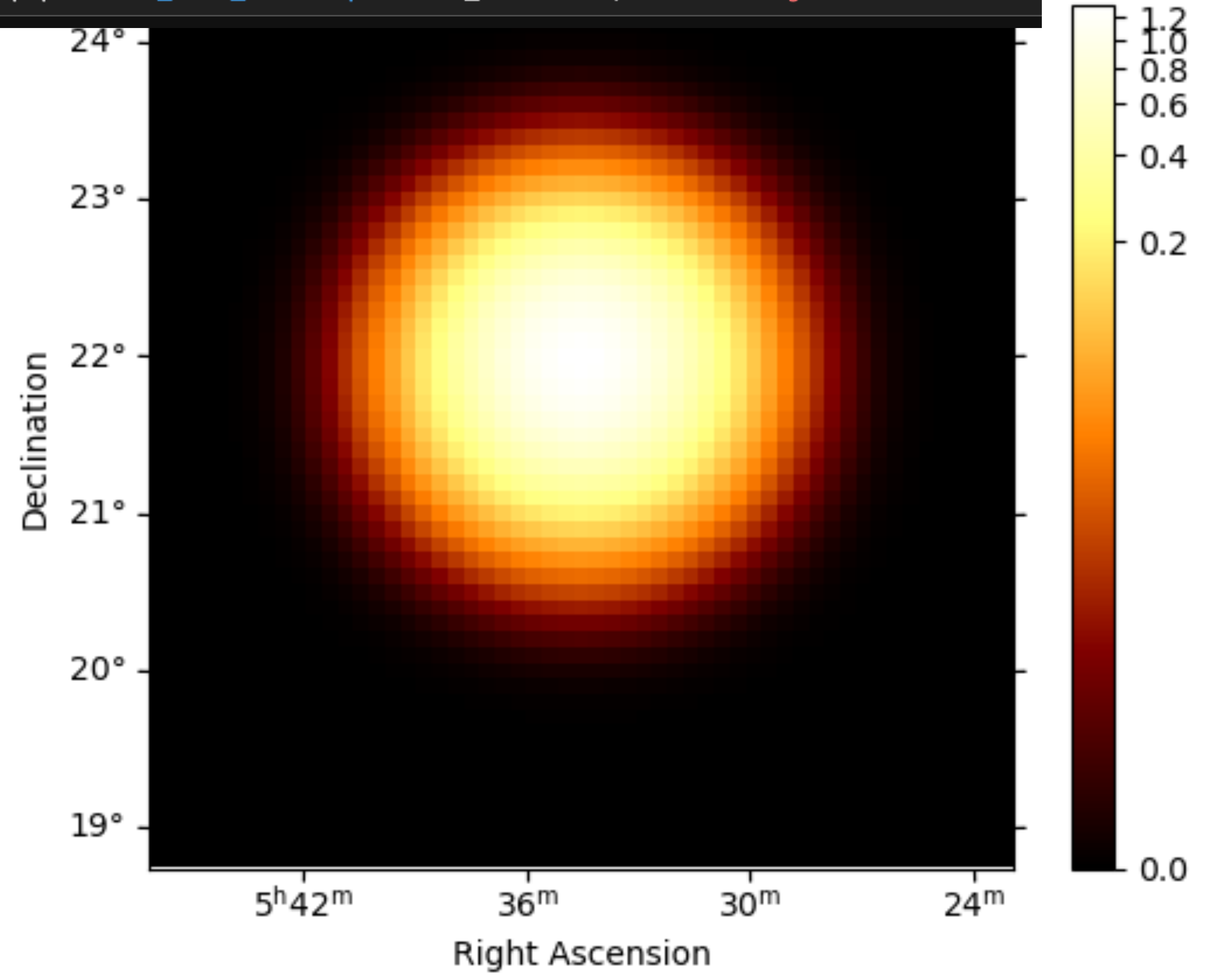

```
mapnpred          = dataset.npred()
mapnpred2         = mapnpred.copy()
mapnpred2.data    = mapnpred.data/ mapnpred.geom.bin_volume().to( "TeV sr" ).value
flux_interp       = mapnpred2.interp_by_coord( events.map_coord( mapnpred2.geom ))/ u.Unit( "TeV sr" )
```

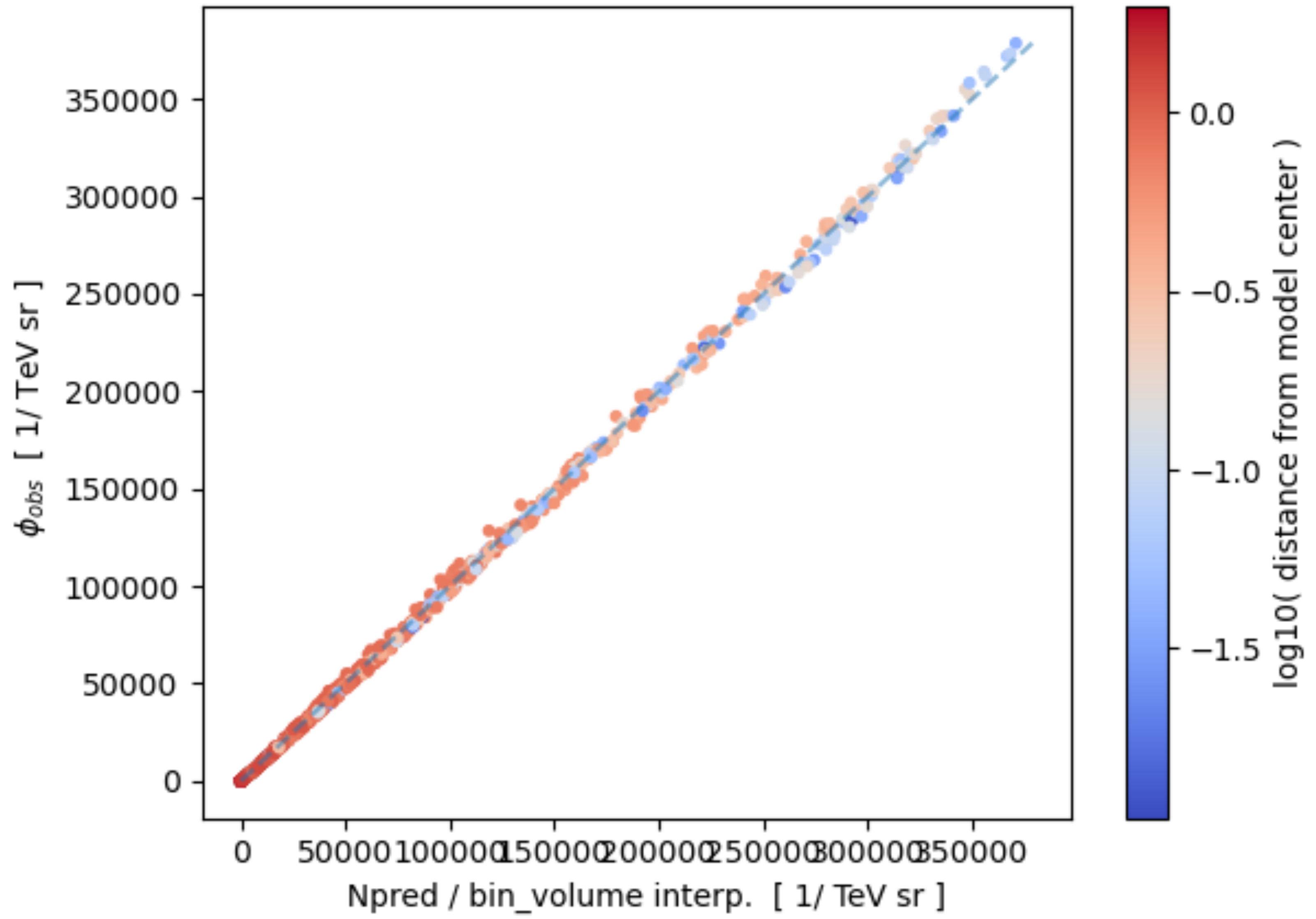


Sigma = 0.5 deg.

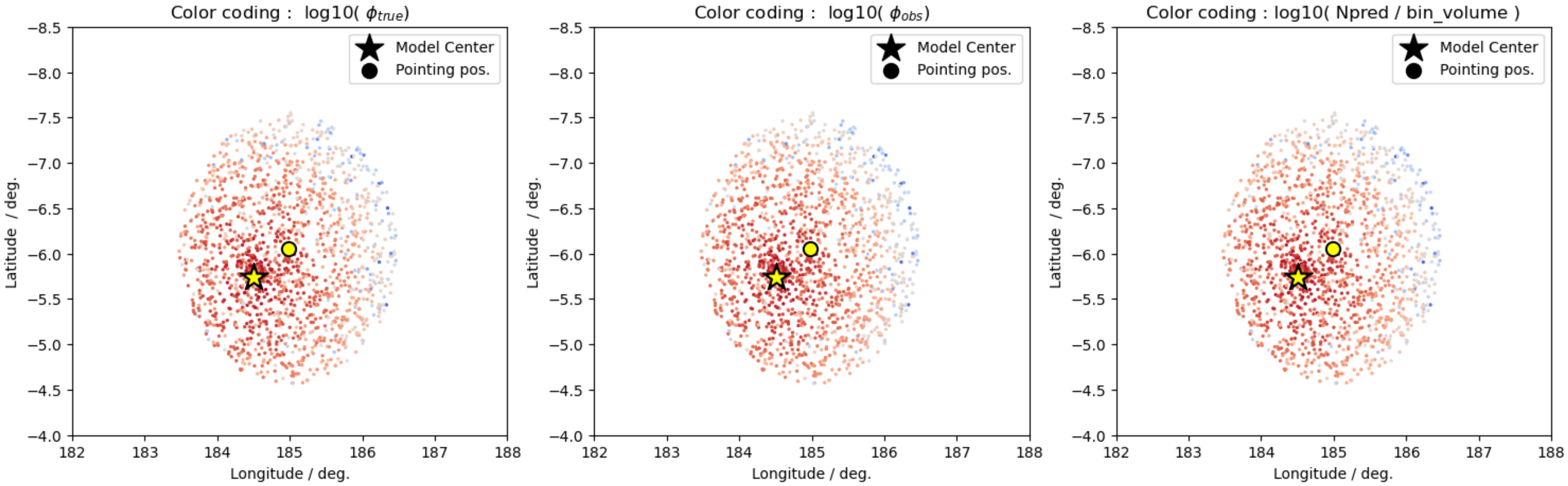
```
model_gauss = SkyModel(
    spatial_model  = GaussianSpatialModel(     lon_0     = "184.557 deg",            lat_0     = "-5.784 deg", sigma = '0.5 deg', frame = 'galactic'),
    spectral_model = LogParabolaSpectralModel( amplitude ='3.5e-11 cm-2 s-1 TeV-1', reference = '1 TeV',      alpha = 1.8,        beta  = 0.4),
    name           = 'crab_model_gauss'
)
```

```
mapnpred       = dataset.npred()
mapnpred       = dataset.evaluators['crab_model_gauss'].compute_npred()

mapnpred.sum_over_axes().plot(add_cbar=True,stretch='log')
```





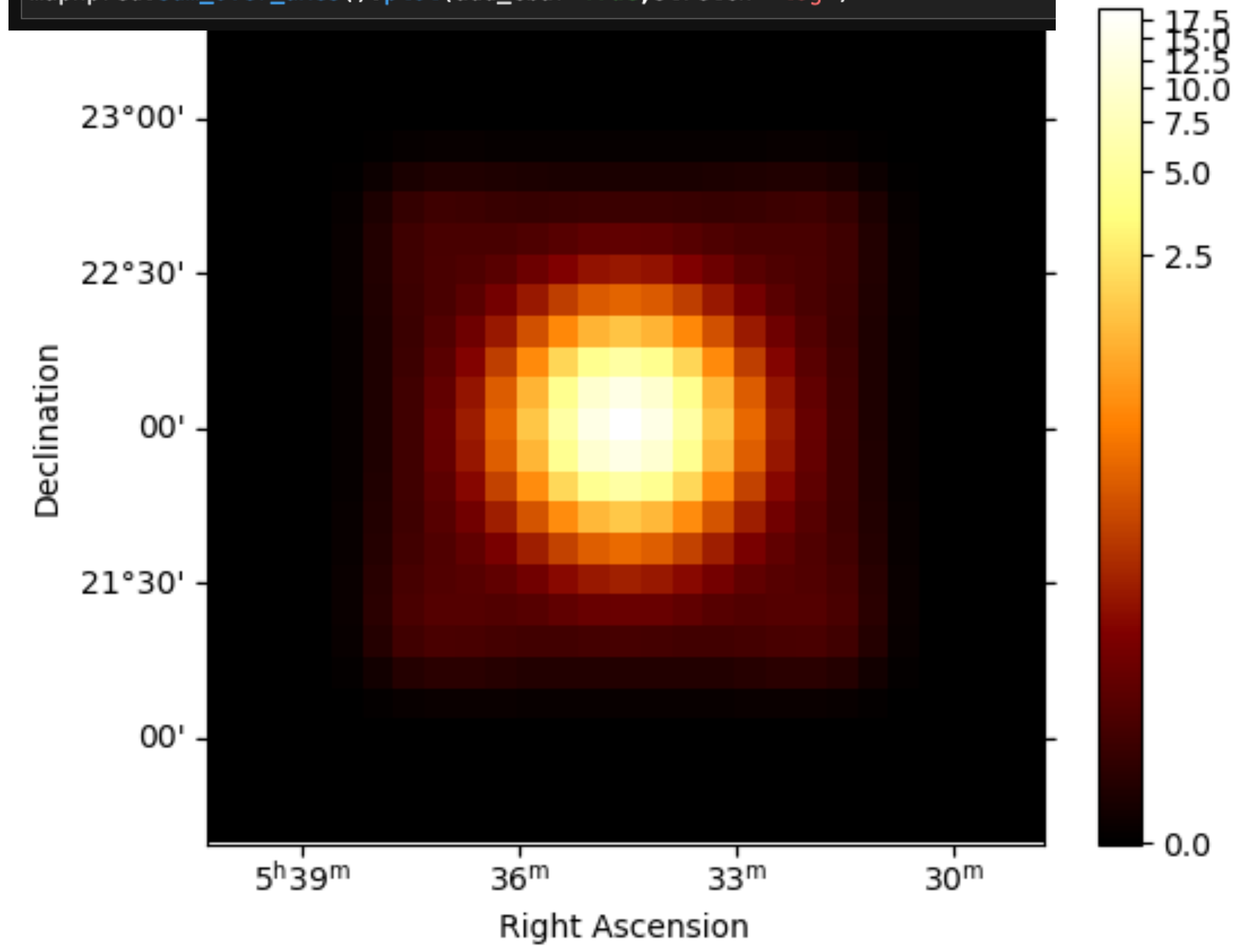

```
mapnpred         = dataset.npred()
mapnpred2        = mapnpred.copy()
mapnpred2.data   = mapnpred.data/ mapnpred.geom.bin_volume().to( "TeV sr" ).value
flux_interp      = mapnpred2.interp_by_coord( events.map_coord( mapnpred2.geom ))/ u.Unit( "TeV sr" )
```

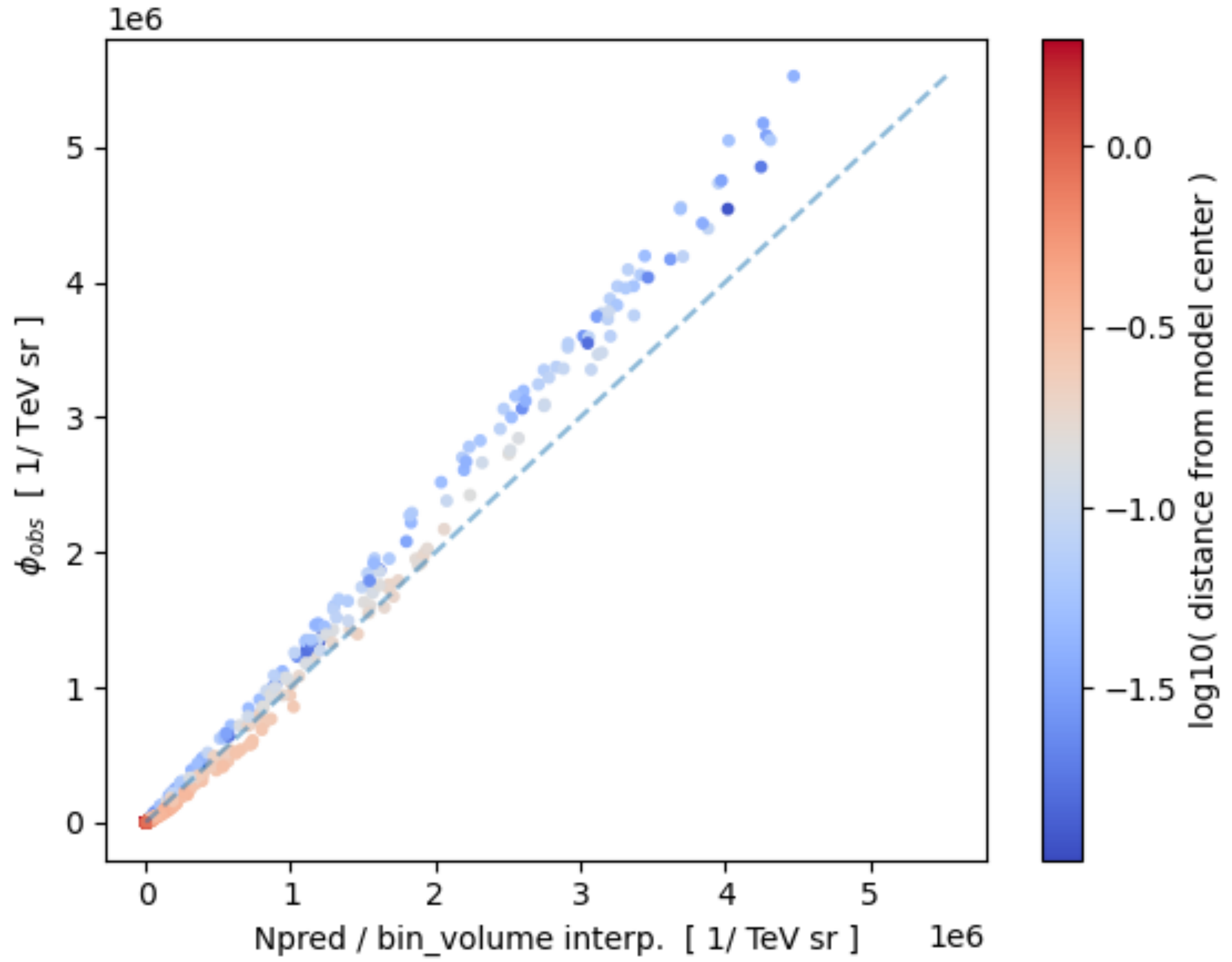


Sigma = 0.1 deg.

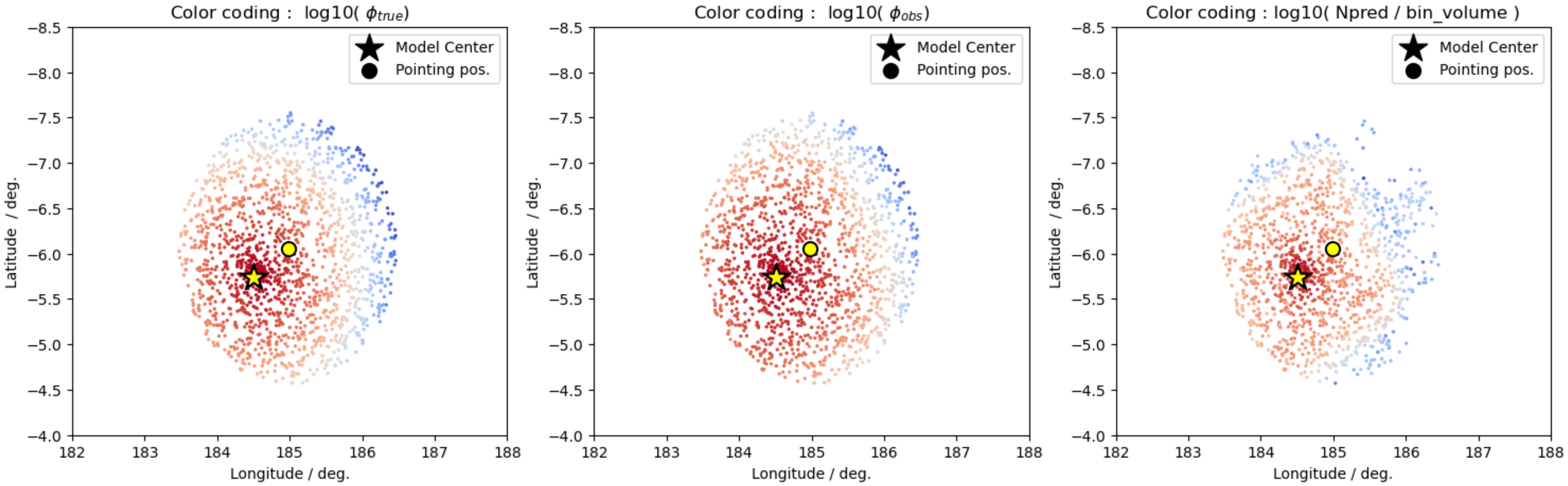
```
model_gauss = SkyModel(
    spatial_model  = GaussianSpatialModel(     lon_0     = "184.557 deg",            lat_0     = "-5.784 deg", sigma = '0.1 deg', frame = 'galactic'),
    spectral_model = LogParabolaSpectralModel( amplitude ='3.5e-11 cm-2 s-1 TeV-1', reference = '1 TeV',      alpha = 1.8,       beta  = 0.4),
    name           = 'crab_model_gauss'
)
```

```
mapnpred        = dataset.npred()
mapnpred        = dataset.evaluators['crab_model_gauss'].compute_npred()

mapnpred.sum_over_axes().plot(add_cbar=True,stretch='log')
```





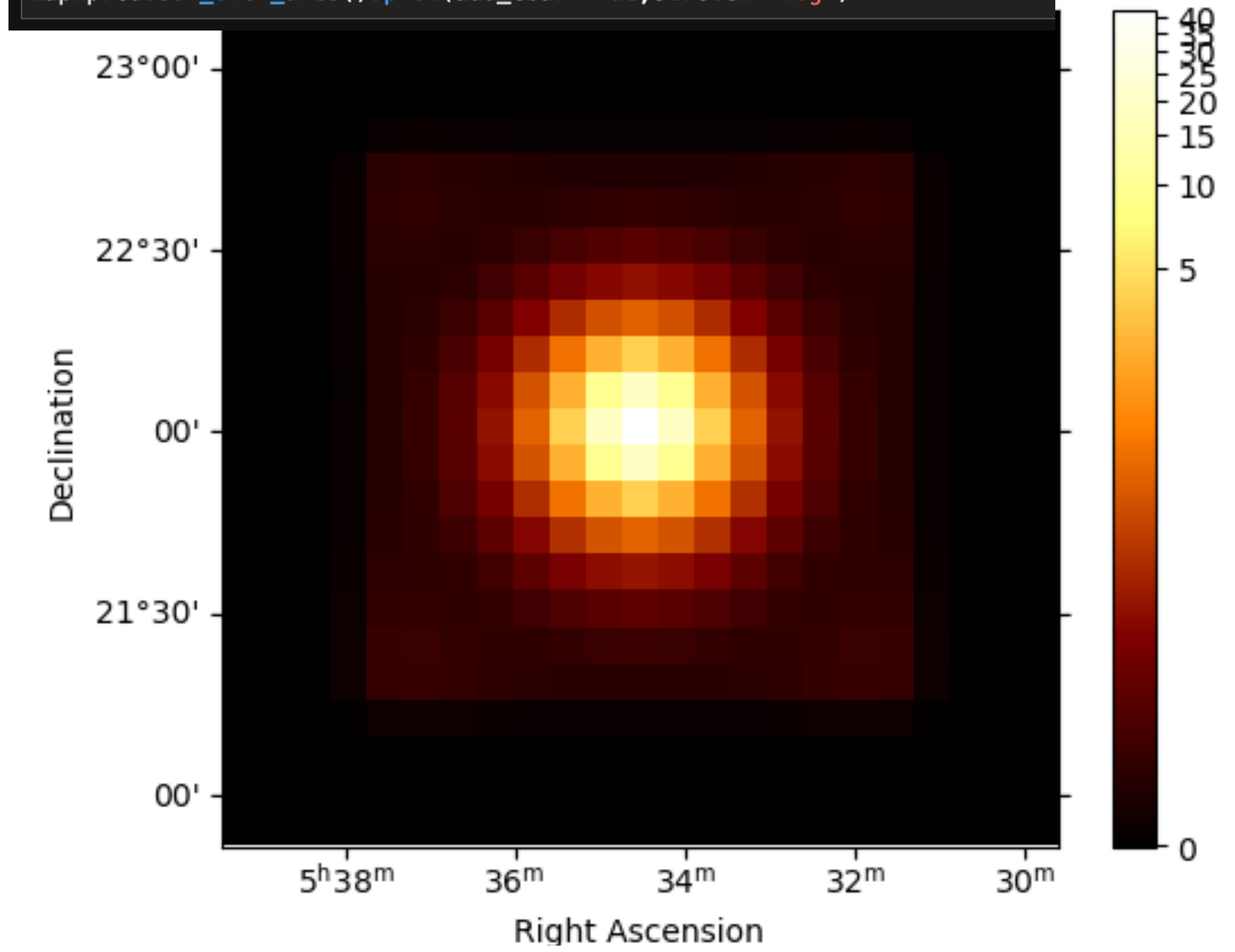

```
mapnpred          = dataset.npred()
mapnpred2         = mapnpred.copy()
mapnpred2.data    = mapnpred.data/ mapnpred.geom.bin_volume().to( "TeV sr" ).value
flux_interp       = mapnpred2.interp_by_coord( events.map_coord( mapnpred2.geom ))/ u.Unit( "TeV sr" )
```

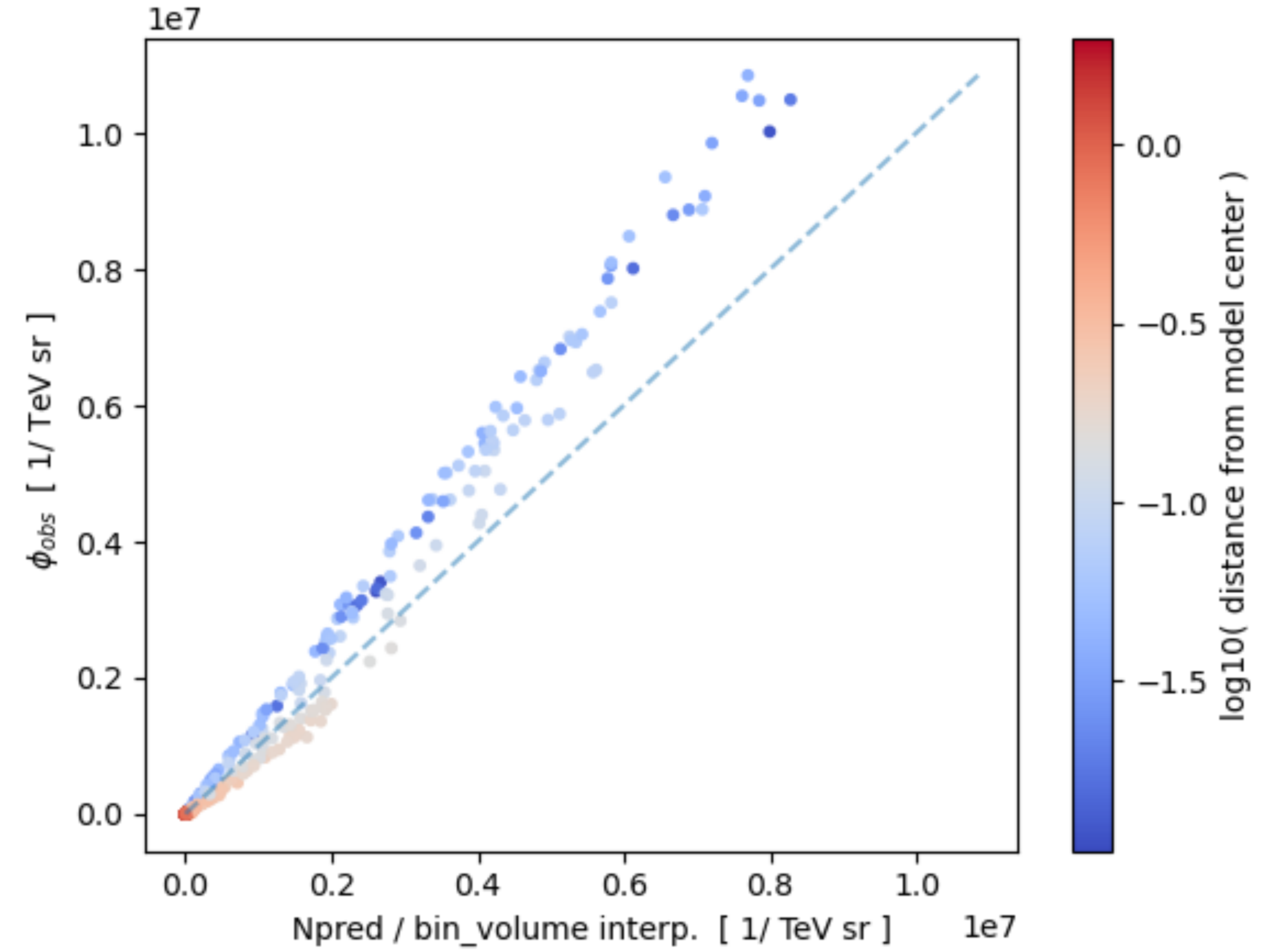


Sigma = 0.05 deg.

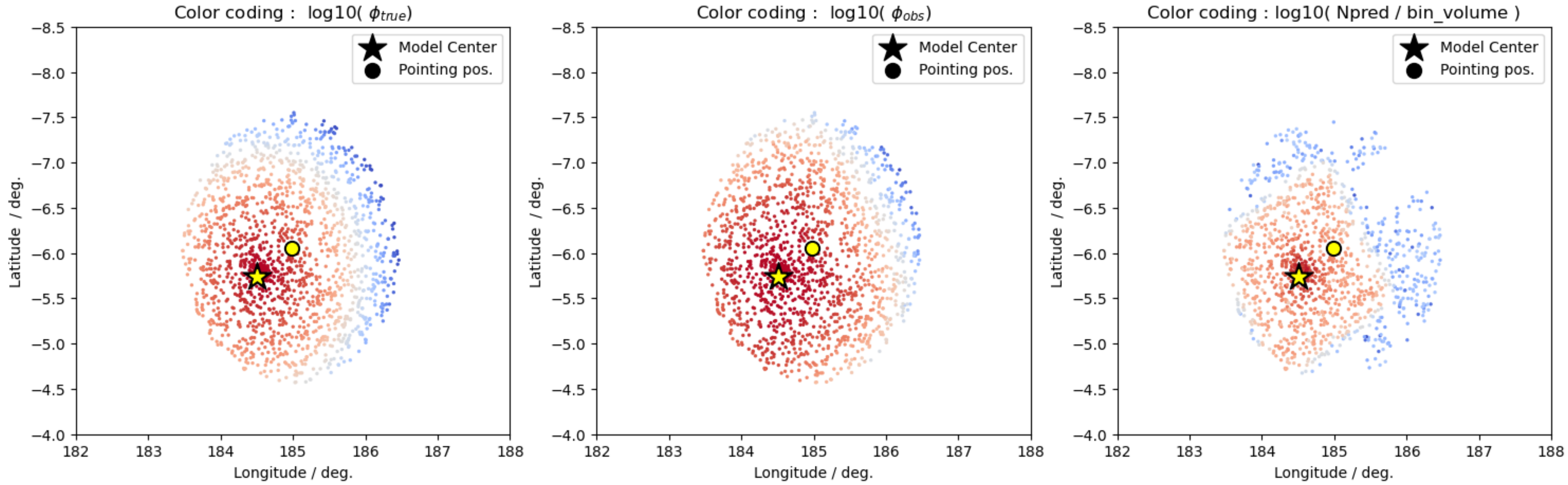
```
model_gauss = SkyModel(
    spatial_model  = GaussianSpatialModel(     lon_0     = "184.557 deg",           lat_0     = "-5.784 deg", sigma = '0.05 deg', frame = 'galactic'),
    spectral_model = LogParabolaSpectralModel( amplitude ='3.5e-11 cm-2 s-1 TeV-1', reference = '1 TeV',      alpha = 1.8,       beta  = 0.4),
    name           = 'crab_model_gauss'
)
```

```
mapnpred      = dataset.npred()
mapnpred      = dataset.evaluators['crab_model_gauss'].compute_npred()

mapnpred.sum_over_axes().plot(add_cbar=True,stretch='log')
```





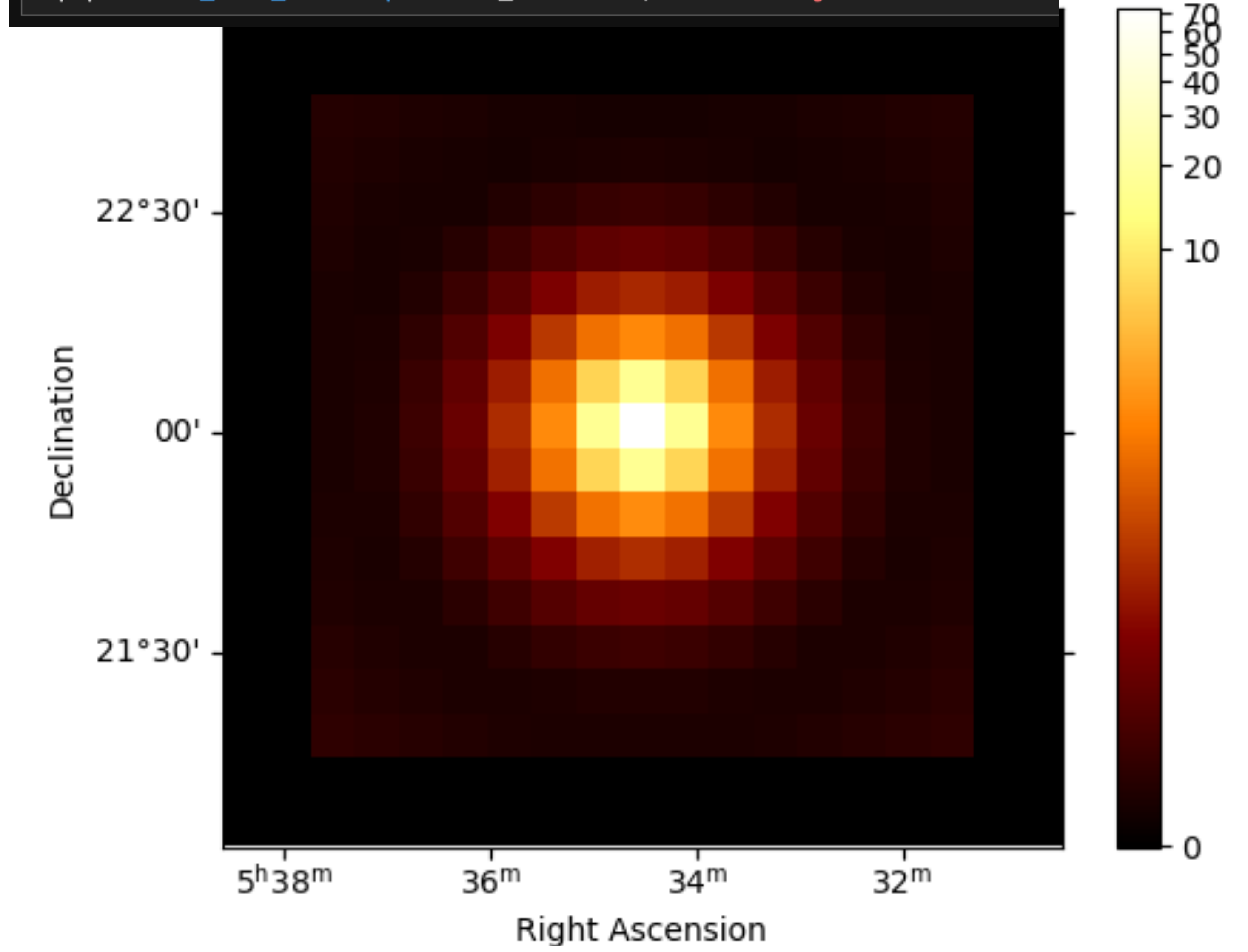

```
mapnpred           = dataset.npred()
mapnpred2          = mapnpred.copy()
mapnpred2.data     = mapnpred.data/ mapnpred.geom.bin_volume().to( "TeV sr" ).value
flux_interp        = mapnpred2.interp_by_coord( events.map_coord( mapnpred2.geom ))/ u.Unit( "TeV sr" )
```

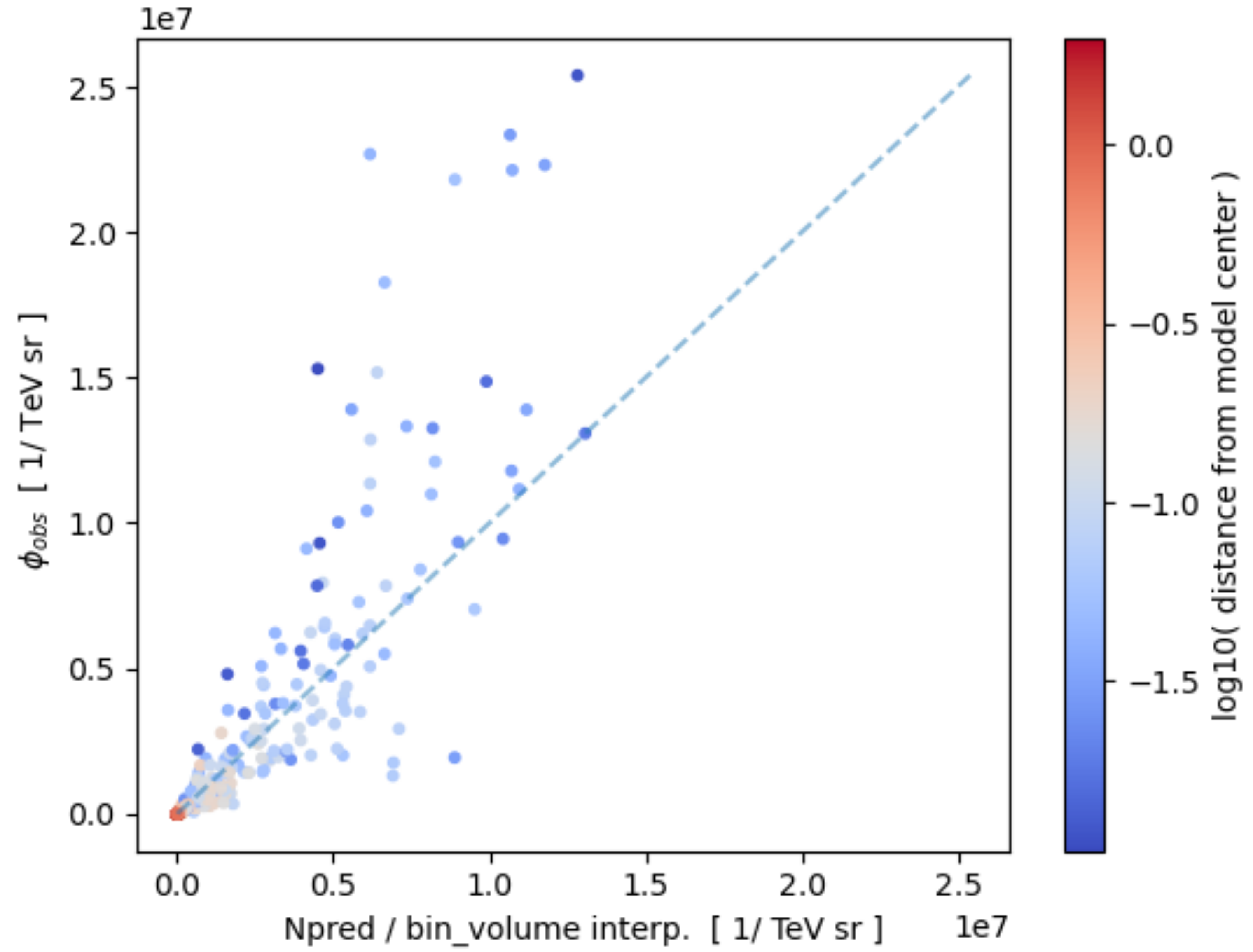


Sigma = 0.01 deg.

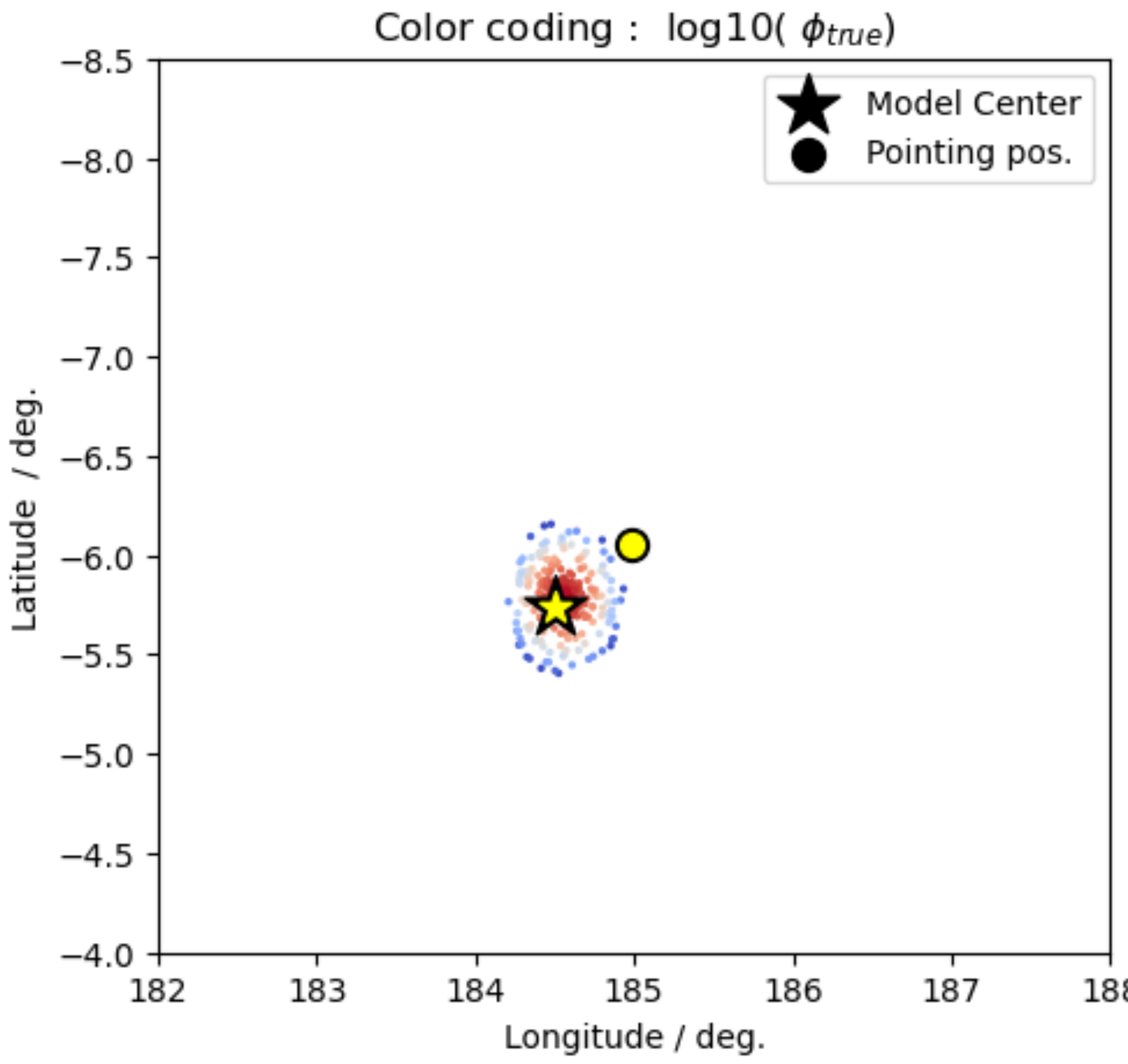
```python
model_gauss = SkyModel(
    spatial_model  = GaussianSpatialModel(     lon_0     = "184.557 deg",          lat_0     = "-5.784 deg", sigma = '0.01 deg', frame = 'galactic'),
    spectral_model = LogParabolaSpectralModel( amplitude ='3.5e-11 cm-2 s-1 TeV-1', reference = '1 TeV',      alpha = 1.8,       beta  = 0.4),
    name           = 'crab_model_gauss'
)
```

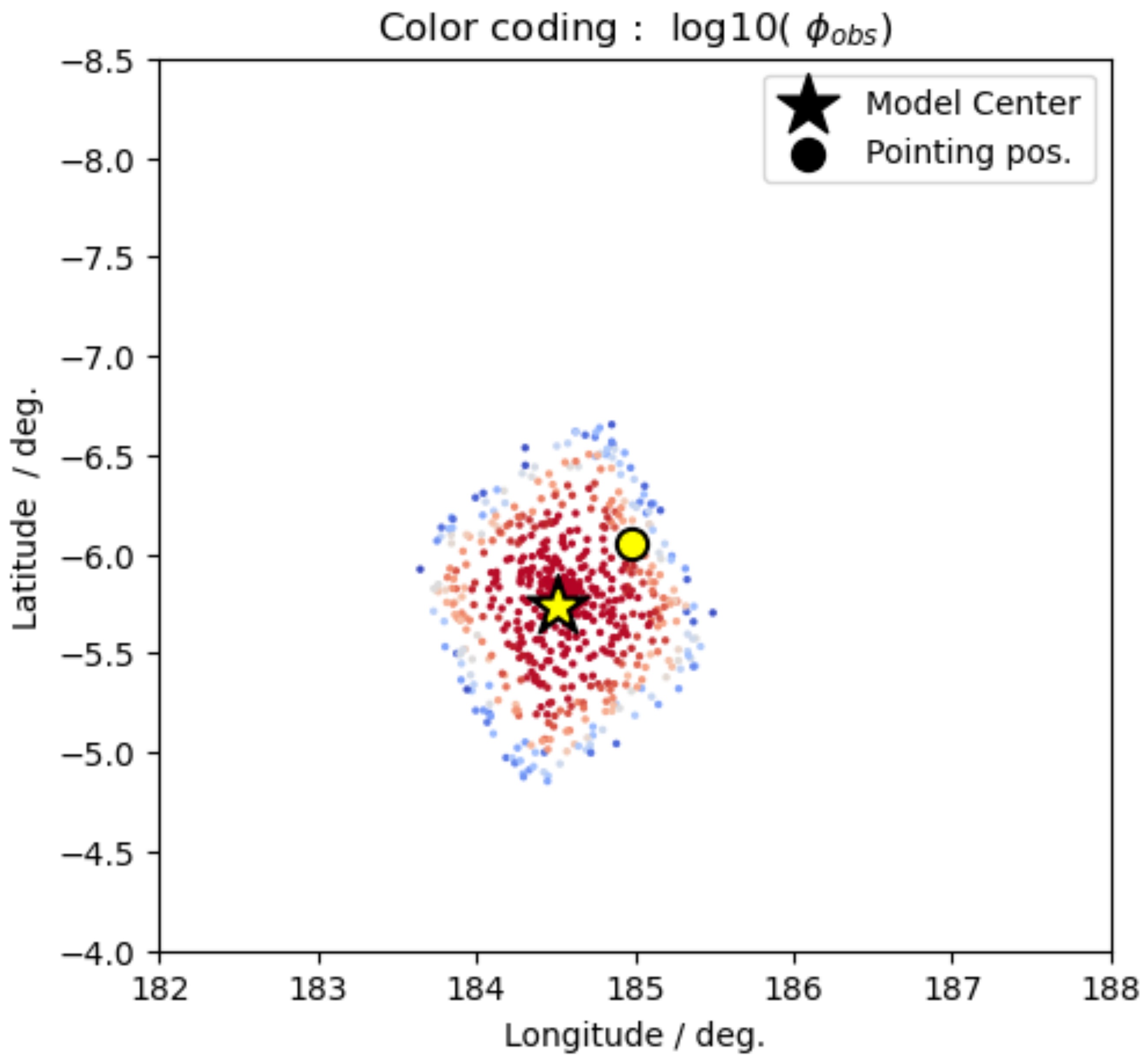
```python
mapnpred       = dataset.npred()
mapnpred       = dataset.evaluators['crab_model_gauss'].compute_npred()

mapnpred.sum_over_axes().plot(add_cbar=True,stretch='log')
```

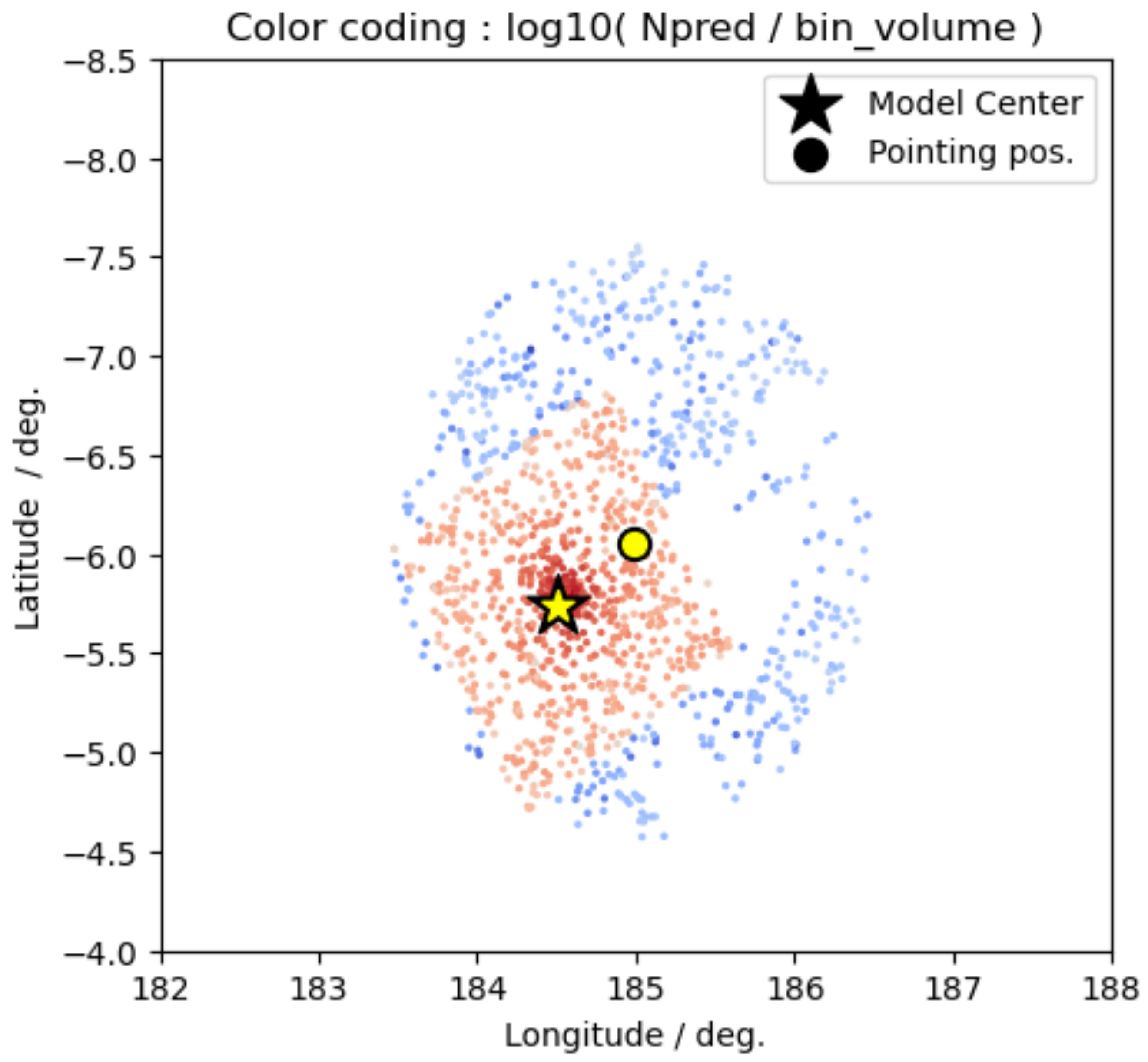





```python
mapnpred        = dataset.npred()
mapnpred2       = mapnpred.copy()
mapnpred2.data  = mapnpred.data/ mapnpred.geom.bin_volume().to( "TeV sr" ).value
flux_interp     = mapnpred2.interp_by_coord( events.map_coord( mapnpred2.geom ))/ u.Unit( "TeV sr" )
```

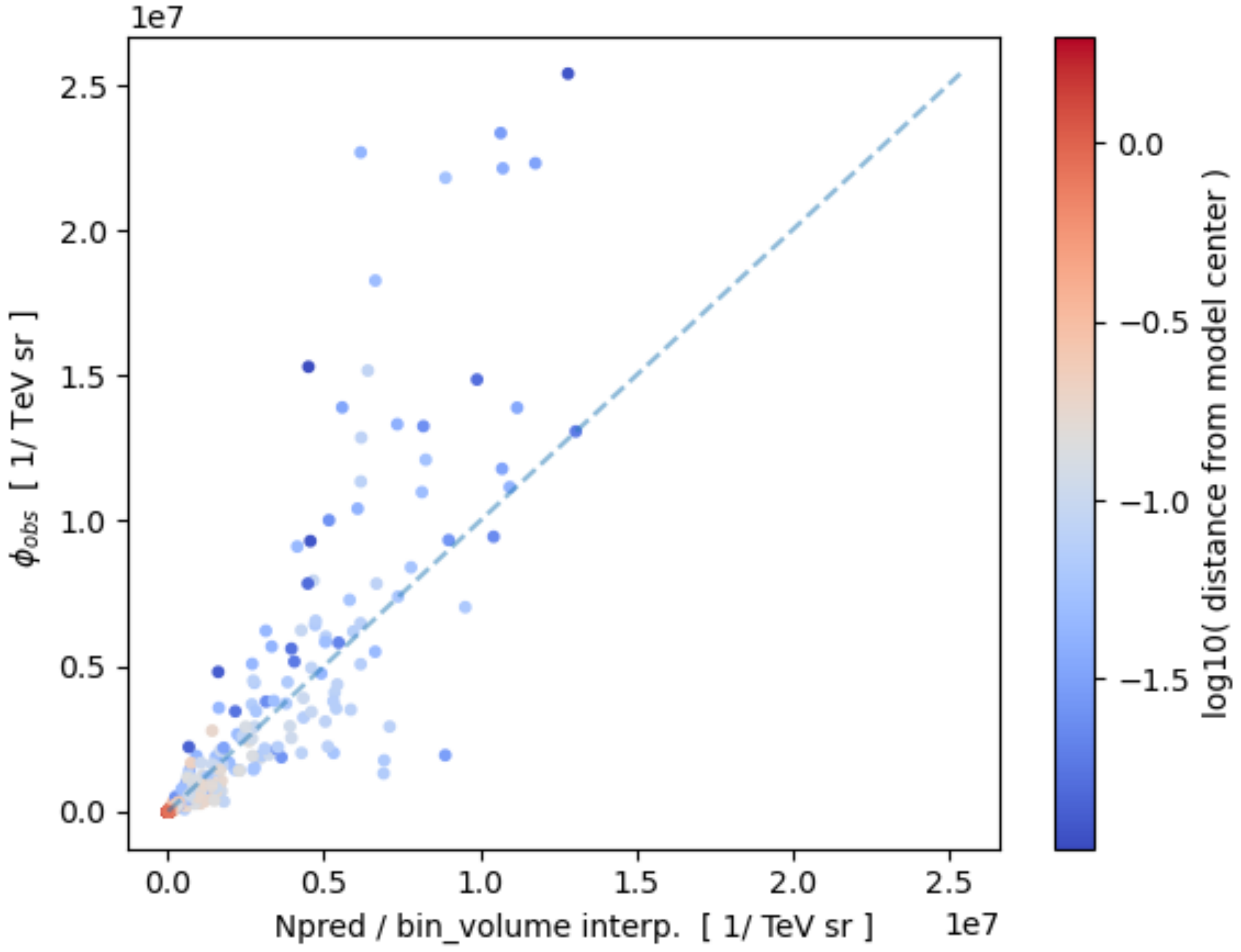






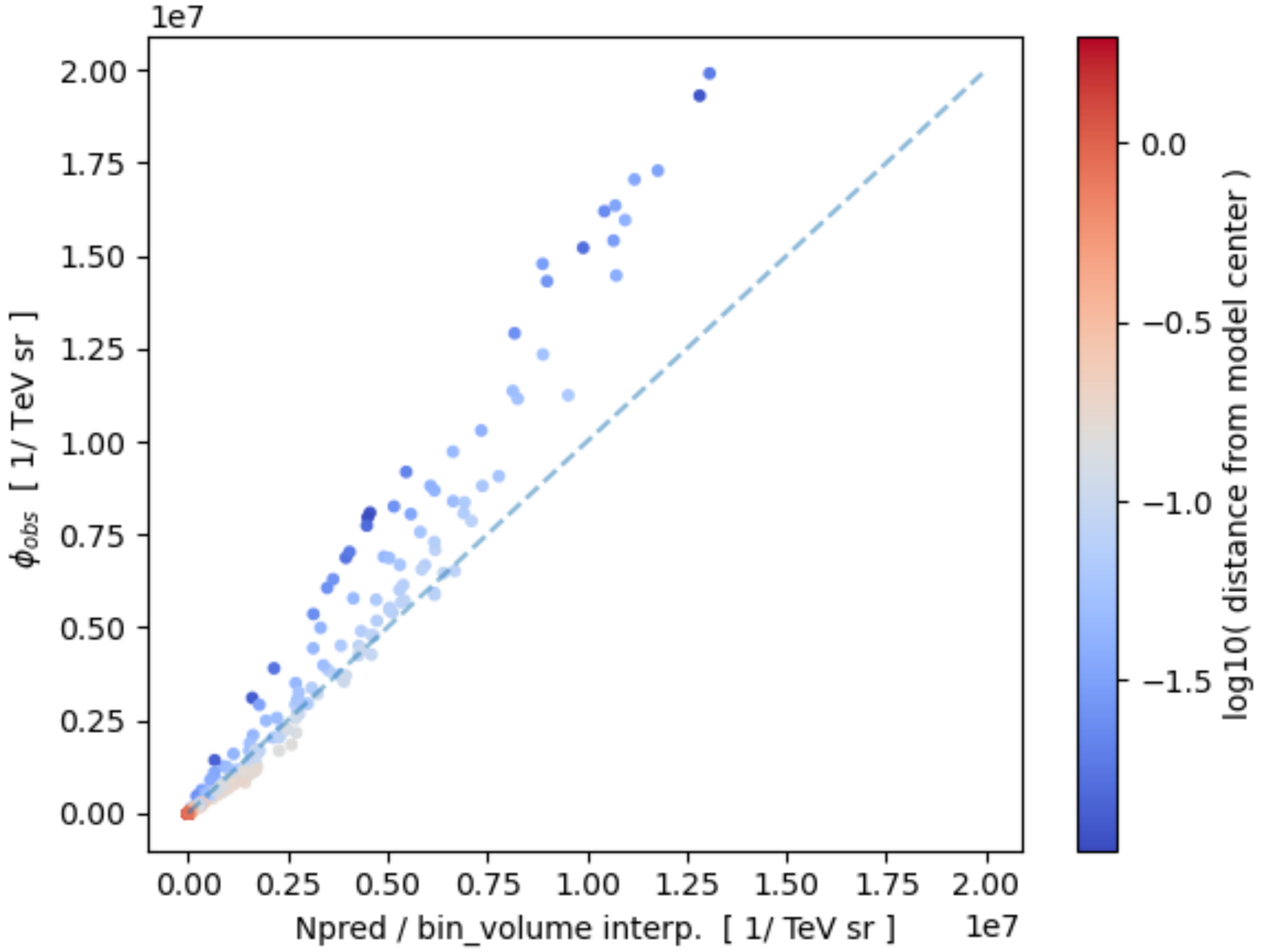
Sigma = 0.01 deg.

$(N_i , N_k , N_l , N_b) = (1665 , 20 , 20 , 20)$



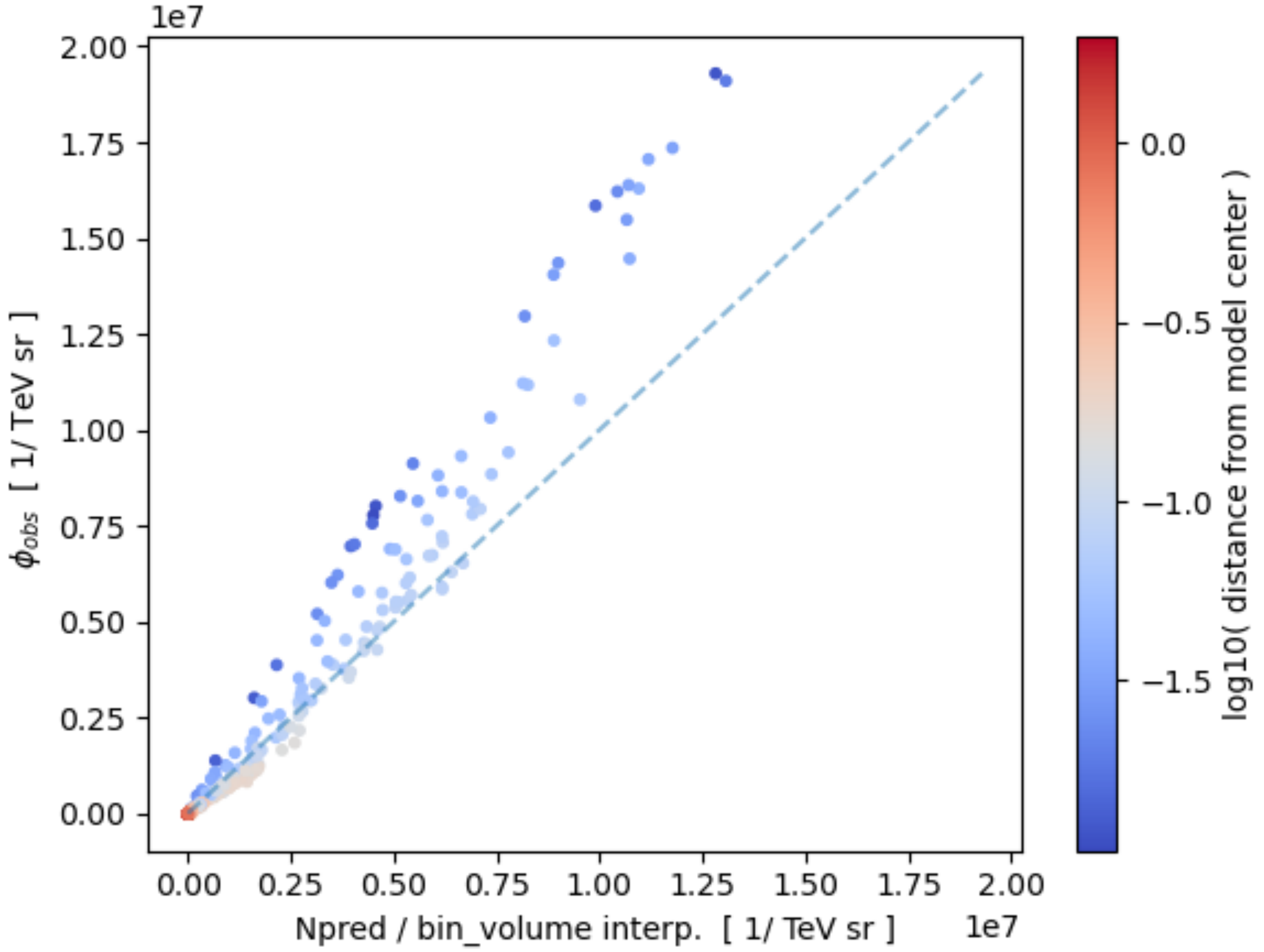
13 s + 0.5 s

$(N_i , N_k , N_l , N_b) = (1665 , 20 , 40 , 40)$



126 s + 9 s

$(N_i , N_k , N_l , N_b) = (1665 , 20 , 80 , 80)$



20 min + 1.5 min

CONCLUSION: when the **source extension** is smaller than the **PSF**, one can get better results by **increasing** the binning in **“true ra_dec”** but this will make the whole analysis much slower.

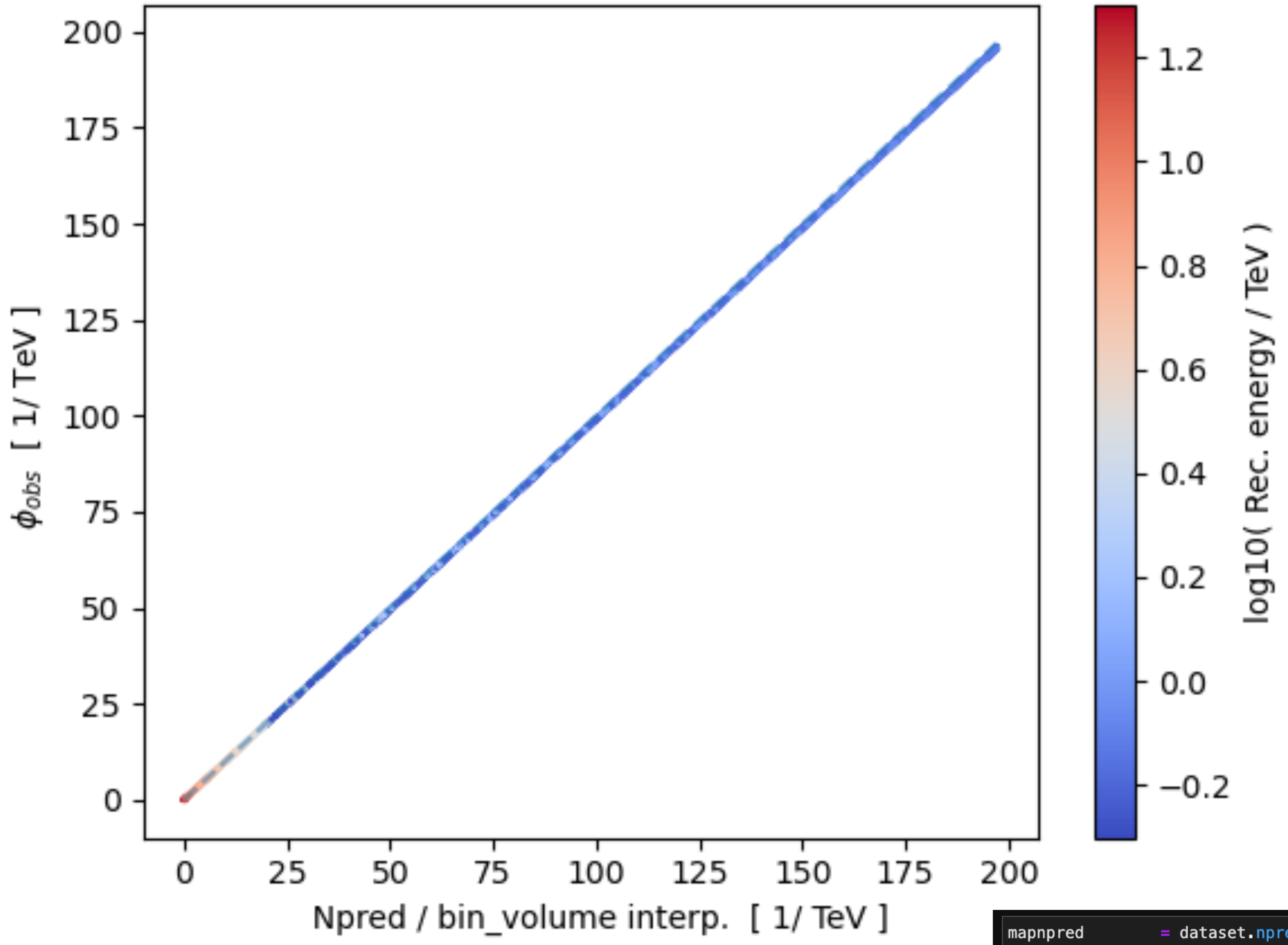
Although, if the source extension is smaller than the PSF, it is worth to only focus on the **energy dimension.**

$$\phi(E_i) = \sum_k \Delta E'_{i,k} \mathrm{D}(E_i | E'_{i,k}, \mathrm{O}) \times \mathrm{A}(E'_{i,k}, \mathrm{O}) \times \phi'(E'_{i,k})$$

O = Offset between model center and pointing

Obs. Flux		Delta Energy		Edisp		Coll. Area		True Flux
(N_i)	$= \sum_{k,l,b}$	(N_i, N_k)	x	(N_i, N_k)	x	(N_i, N_k)	x	(N_i, N_k)
1 / TeV		TeV		1 / TeV		s x m^2		1 / (s x m^2 x TeV)

For the 1D case the whole "matrix multiplication" took only 50 ms



1-Dimensional Case

```
mapnpred        = dataset.npred()
mapnpred2       = mapnpred.copy()
omega           = on_region.radius**2 * np.pi
mapnpred2.data  = mapnpred.data/ mapnpred.geom.bin_volume().to( "TeV sr" ).value *  omega.to( 'sr').value
flux_interp     = mapnpred2.interp_by_coord( events.map_coord( mapnpred2.geom ))/ u.Unit( "TeV"  )
```