The Crab Nebula seen through the CTA observatory

Enrique Mestre

Project Description

- The Crab pulsar and its nebula is the most studied source of the Rotation Powered Pulsars (RPPs) and their nebulae, fast rotating neutron stars within the remnant of the supernova explosion (SNRe) that gave birth to the pulsars powering them.
- They are also the largest known population of gamma-ray sources in our galaxy
- Efficient particles accelerators
- Present variability in time scales from years to hours (flares).

Project Description

We are interested in:

- The Crab Nebula: Spectral shape + Morphology + Flares

- The Crab Pulsar: Synchrotron-curvature spectrum + Very-high-energy tail + Pulsed emission

Project Description

We are interested in:

- The Crab Nebula: Spectral shape + Morphology + **Flares**

- The Crab Pulsar: Synchrotron-curvature spectrum + Very-high-energy tail + Pulsed emission

Project Description

We are interested in:

- The Crab Nebula: Spectral shape + Morphology + **Flares**
- The Crab Pulsar: **Synchrotron-curvature spectrum** + **Very-high-energy** tail + **Pulsed emission**

Methodology

- Why do we need a cube: two objects in the same position with different morphology and spectral characteristics
- Description of the simulations:

Cube Generator:
- Source models
- Convolution with CTA IRFs
- Background cube
- PSF convolution

- Projection X, Y: Sherpa Fitting
- Projection E: SpectrumFit

- 3D Fitting (in progress)
- Sherpa – but no PSF yet

Spectral Analysis: General

- Making the cube

1. Nebula and pulsar models in yaml files (binning, pointing, spectrum model…):
   - Waiting for the xml implementation (generalisation to n sources)

```
def read_config(filename):
    with open(filename) as fh:
        config = yaml.load(fh)
    config['model']['prefactor'] = float(config['model']['prefactor'])
    return config

config = read_config('config.yaml')
```

Spectral Analysis: General

- Making the cube

  2. Get CTA response functions:

```
def get_irfs(config):
    filename = 'irf_file.fits'

    offset = Angle(config['selection']['offset_fov'] * u.deg)

    psf_fov = EnergyDependentMultiGaussPSF.read(filename, hdu='POINT SPREAD FUNCTION')
    psf = psf_fov.to_energy_dependent_table_psf(theta=offset)

    aeff = EffectiveAreaTable2D.read(filename, hdu='EFFECTIVE AREA')

    edisp_fov = EnergyDispersion2D.read(filename, hdu='ENERGY DISPERSION')
    table = fits.open(filename)['BACKGROUND']
    table.columns.change_name(str('BGD'), str('Bgd'))
    table.header['TUNIT7'] = '1 / (MeV s sr)'

    bkg = Background3D.read(filename, hdu='BACKGROUND')

    return dict(psf=psf, aeff=aeff, edisp=edisp_fov, bkg=bkg, offset = offset)

irfs = get_irfs(config)
```

Spectral Analysis: General

- Making the cube

3. Make a CombineModel3D:

```
def get_model_gammapy(config):
   if config['model']['template'] == 'Shell2D':
        spatial_model = Shell2D(
            amplitude=1,
            x_0=config['model']['ra'],
            y_0=config['model']['dec'],
            r_in=config['model']['rin'],
            width=config['model']['width'],
            # Note: for now we need spatial models that are normalised
            # to integrate to 1 or results will be incorrect!!!
            normed=True,
        )

   if config['model']['spectrum'] == 'pl':
        spectral_model = PowerLaw(
            amplitude=config['model']['prefactor'] * u.Unit('cm-2 s-1 TeV-1'),
            index=config['model']['index'],
            reference=config['model']['pivot_energy'] * u.Unit('TeV'),
        )
   return CombinedModel3D(
        spatial_model=spatial_model,
        spectral_model=spectral_model,
        )

   model = get_model_gammapy(config)
```

Spectral Analysis: General

- Making the cube

4. Make reference cubes:

```
def make_ref_cube(config):
    WCS_SPEC = {
        'nxpix': config['binning']['nxpix'],
        'nypix': config['binning']['nypix'],
        'binsz': config['binning']['binsz'],
        'xref': config['pointing']['ra'],
        'yref': config['pointing']['dec'],
        'proj': config['binning']['proj'],
        'coordsys': config['binning']['coordsys'],
    }

    # define reconstructed energy binning
    ENERGY_SPEC = {
        'mode': 'edges',
        'enumbins': config['binning']['enumbins'],
        'emin': config['selection']['emin'],
        'emax': config['selection']['emax'],
        'eunit': 'TeV',
    }

    CUBE_SPEC = {}
    CUBE_SPEC.update(WCS_SPEC)
    CUBE_SPEC.update(ENERGY_SPEC)
    cube = SkyCube.empty(**CUBE_SPEC)
    return cube

ref_cube = make_ref_cube(config)
```

Spectral Analysis: General

- Making the cube

  5. Evaluate the model in the reference cubes and sum the fluxes:

```
flux_cube = model.evaluate_cube(ref_cube)
flux_cube2 = model2.evaluate_cube(ref_cube2)

def compute_sum_cube(flux_cube, flux_cube2, config):
    ebin = flux_cube.energies(mode="edges")
    ebounds = EnergyBounds(ebin)
    nflux_cube_sum = make_ref_cube(config)

    for idx in range(len(ebounds) - 1):
        npred1 = flux_cube.sky_image_idx(idx)
        npred2 = flux_cube2.sky_image_idx(idx)

        nflux_sum = u.Quantity(npred1.data.value + npred2.data.value,'1 / (cm2 s sr TeV)')
        nflux_cube_sum.data[idx] = nflux_sum.value

    return nflux_cube_sum

flux_sum=compute_sum_cube(flux_cube, flux_cube2, config)
```

Spectral Analysis: General

- Making the cube

6. Make predicted counts cube

```python
def compute_npred_cube(flux_cube, exposure_cube, ebounds, config, irfs, integral_resolution=10):

    # Make an empty cube with the requested energy binning
    sky_geom = exposure_cube.sky_image_ref
    energies = EnergyBounds(ebounds)
    npred_cube = SkyCube.empty_like(sky_geom, energies=energies, unit='', fill=np.nan)

    # Process and fill one energy bin at a time
    for idx in range(len(ebounds) - 1):
        emin, emax = ebounds[idx: idx + 2]
        ecenter = np.sqrt(emin * emax)
        flux = flux_cube.sky_image_integral(emin, emax, interpolation='linear', nbins=integral_resolution)
        exposure = exposure_cube.sky_image(ecenter, interpolation='linear')
        solid_angle = exposure.solid_angle()
        flux.data.value[np.isnan(flux.data.value)]=0
        exposure.data.value[np.isnan(exposure.data.value)]=0
        npred = flux.data.value * u.Unit('1 / (cm2 s sr)') * exposure.data * solid_angle
        npred_cube.data[idx] = npred.to('')

    # Apply EnergyDispersion
    edisp = irfs['edisp']
    offset = irfs['offset']

    edisp_idx = edisp.to_energy_dispersion(offset=offset, e_reco = ebounds, e_true = ebounds)

    for pos_x in range(npred_cube.data.shape[1]):
        for pos_y in range(npred_cube.data.shape[2]):
            npred_pos = npred_cube.data[0:len(ebounds) - 1,pos_x,pos_y]
            if npred_pos.sum() != 0:
               for idx in range(len(ebounds) - 1):
                   npred_cube.data[idx][pos_x][pos_y] = np.dot(npred_pos, edisp_idx.data.data[idx])

    return npred_cube
```

Spectral Analysis: General

- Making the cube

   7. Compute the on and off events and the excess

```
def compute_nexcess_cube(npred_cube, livetime, pointing, offset_max, bkg_rate, config):

    ebin = npred_cube.energies(mode="edges")
    ebounds = EnergyBounds(ebin)
    nexcess_cube = make_ref_cube(config)
    non_cube = make_ref_cube(config)
    noff_cube = make_ref_cube(config)

    # Compute two background cubes
    nbkg1_cube =  make_background_cube(pointing = pointing, obstime = livetime, bkg = bkg_rate, ref_cube=npred_cube, offset_max = offset_max)
    nbkg2_cube =  make_background_cube(pointing = pointing, obstime = livetime, bkg = bkg_rate, ref_cube=npred_cube, offset_max = offset_max)

    # For each energy bin, I need to obtain the correct background rate (two, one for the on and one for the off)
    for idx in range(len(ebounds) - 1):
        emin, emax = ebounds[idx: idx + 2]
        ecenter = np.sqrt(emin * emax)

        npred = npred_cube.sky_image_idx(idx)
        npred.unit = u.Unit('TeV')
        solid_angle = npred.solid_angle()
        npred.data.value[np.isnan(npred.data.value)]=0.

        nbkg1_ebin = nbkg1_cube.data[idx]
        nbkg2_ebin = nbkg2_cube.data[idx]

        n_on = np.random.poisson(npred.data) + np.random.poisson(abs(nbkg1_ebin))
        n_off = np.random.poisson(abs(nbkg2_ebin))
        nexcess = n_on - n_off
        nexcess_cube.data[idx] = nexcess
        non_cube.data[idx] = n_on
        noff_cube.data[idx] = n_off

    return nexcess_cube, non_cube, noff_cube
```

Spectral Analysis: General

- Making the cube

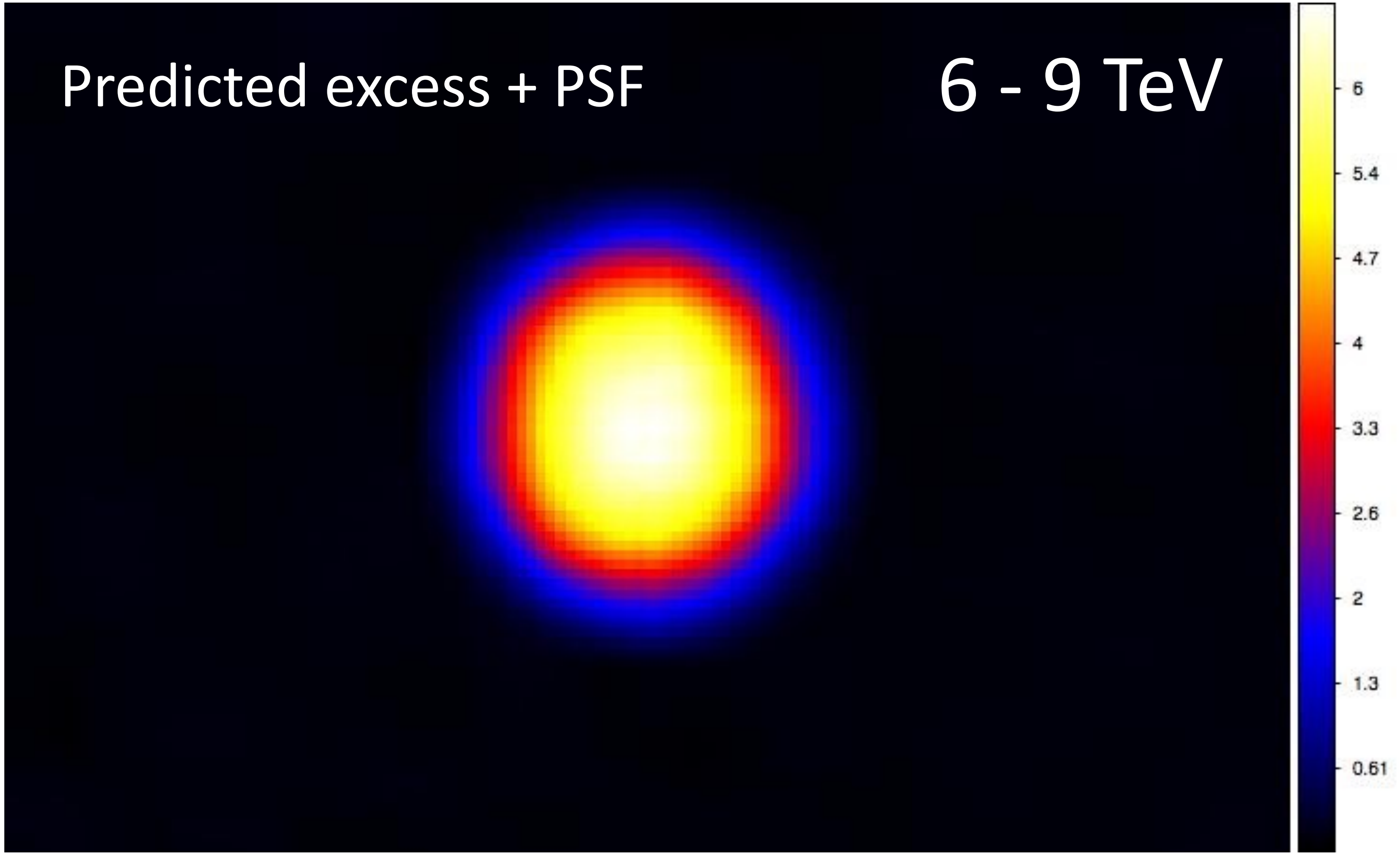
    8. Apply PSF convolution and write the cubes

```
kernels = []

psf_table = psf_fromfits('irf_file.fits')
energ_lo = psf_table[0].value
sigmas = psf_table[3]
energ_array = nexcess_cube.energies('edges')

s = np.argmin(np.abs(irfs['offset'].value - psf_table[2].value))

for i in range(nexcess_cube.data.shape[0]):
    v = np.argmin(np.abs(energ_array[i].value - energ_lo))
    kernels.append(irfs['psf'].kernels(nexcess_cube,Angle(sigmas[0][s][v] * u.deg))[i])

# Apply kernels convolution
nexcess_cube_convolved = nexcess_cube.convolve(kernels)
noff_cube_convolved = noff_cube.convolve(kernels)
non_cube_convolved = non_cube.convolve(kernels)

nexcess_cube_convolved.write('nexcess_cube_convolved.fits.gz', overwrite = True)
noff_cube_convolved.write('noff_cube_convolved.fits.gz', overwrite=True)
non_cube_convolved.write('non_cube_convolved.fits.gz', overwrite = True)
```

Spectral Analysis: General

- Example of cube. Excess events cube convolved with the PSF :

- Model:
  - MAGIC Log-Parabola
  - 2D sphere of 0.1 deg
  - Pulsar index of 2.9
  - Point-like pulsar
  - Time of 100 hours

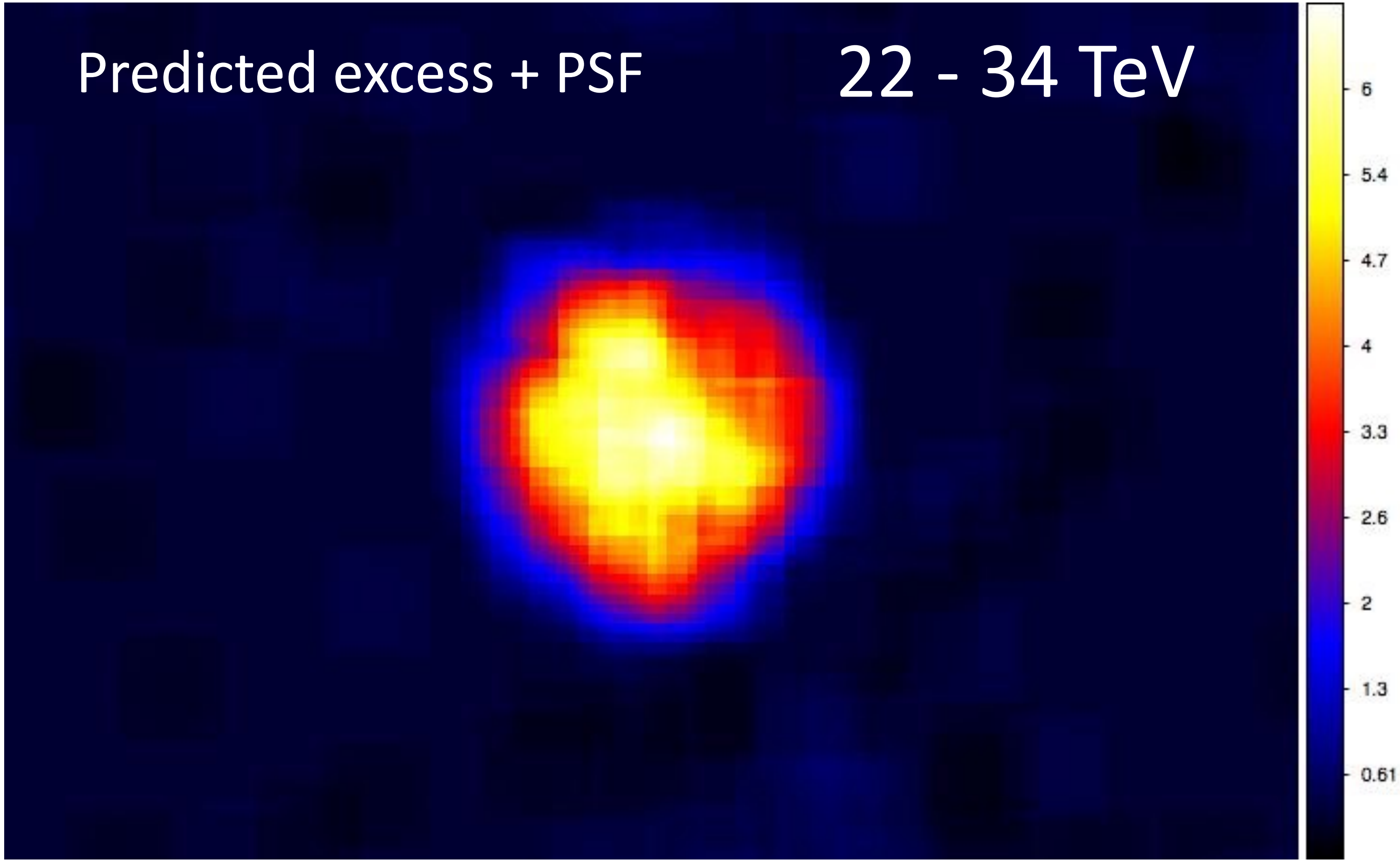


Spectral Analysis: General

- Example of cube. Excess events cube convolved with the PSF :

Same cube at higher energies:

Effect of the PSF energy dependence + on events spatial fluctuations

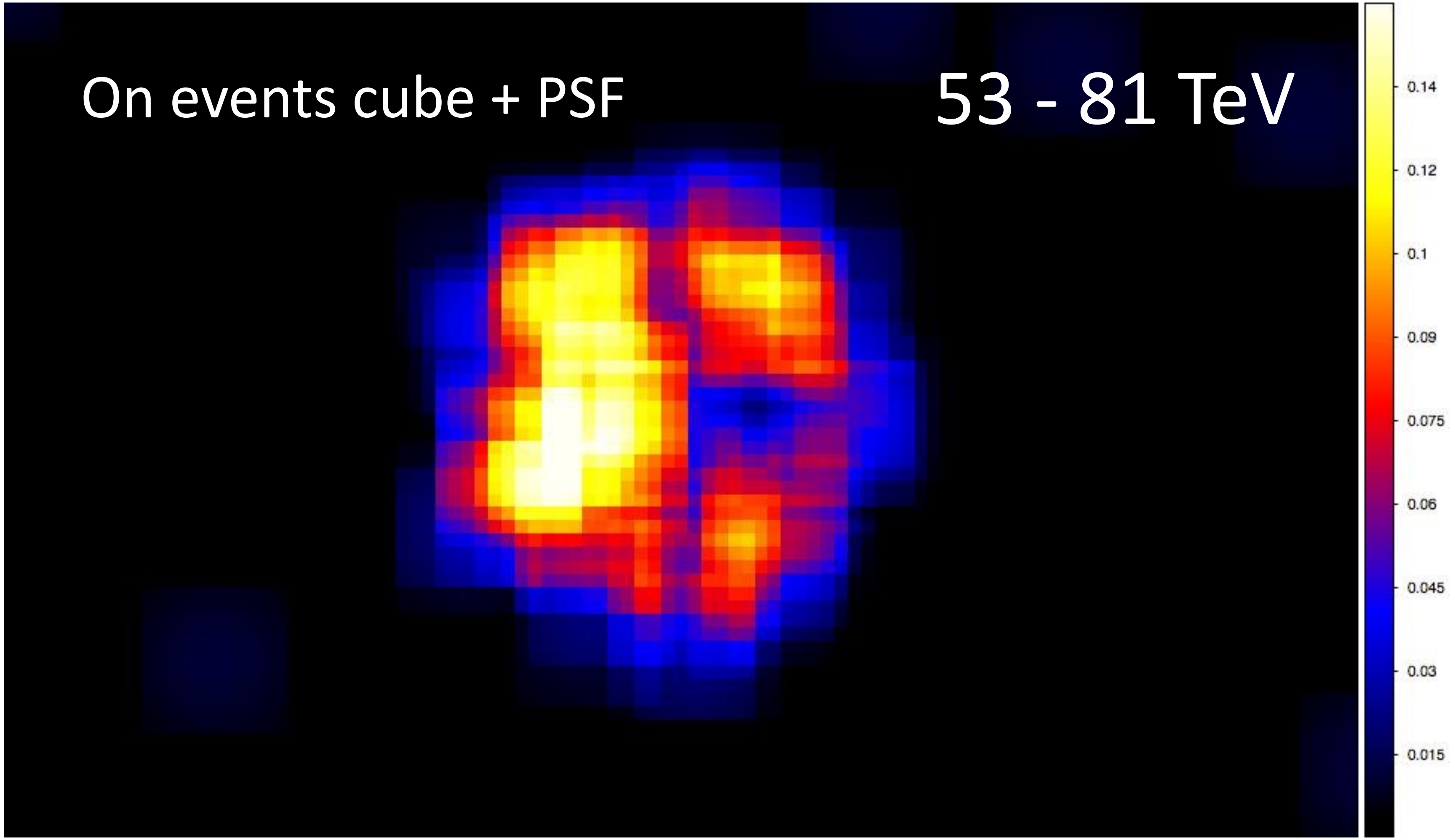


Spectral Analysis: General

- Example of cube. On and off events maps convolved with PSF :

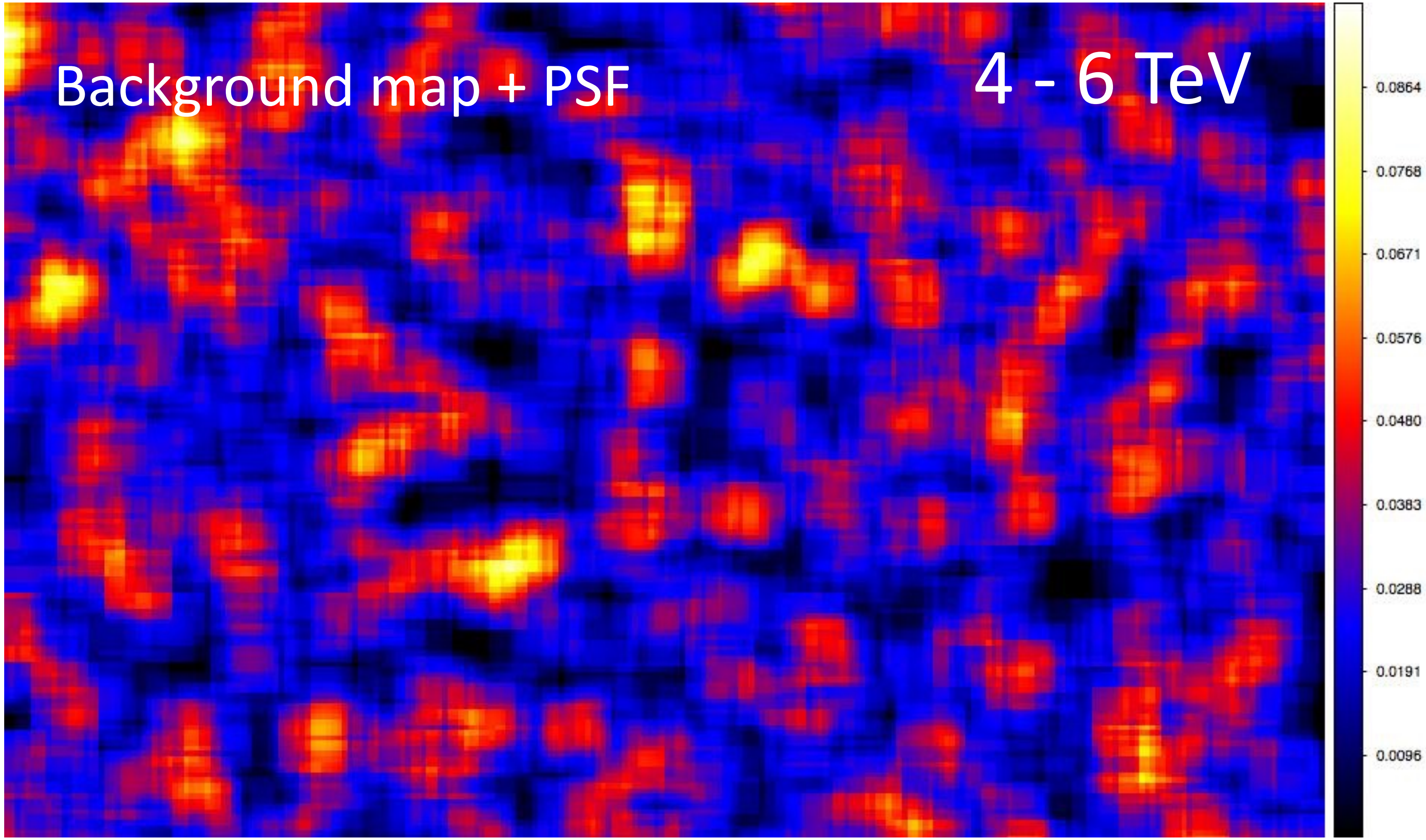
Same cube at higher energies:

Effect of the PSF energy dependence + on events spatial fluctuations



Spectral Analysis: General

- Example of cube. On and off events maps convolved with PSF :



Spectral Analysis: General

- For the moment we project the cube along the E axis to fit the spectrum, we can: Switch the pulsar off (nebula + background) or switch the nebula off (pulsar + background), so for example we take the nebula as part of the background cube in order to fit the pulsar spectrum.

```
filename_on = 'non_cube_convolved.fits.gz'

cube_on = SkyCube.read(filename_on)

circ_filename_off = 'noff_withneb_cube_convolved.fits.gz'

circ_cube_off = SkyCube.read(circ_filename_off)
```

Spectral Analysis: General

1. Define on and off regions by energy bins.

```
filename_on = 'non_cube_convolved.fits.gz' # non_cube_convolved.fits

cube_on = SkyCube.read(filename_on)

circ_filename_off = 'noff_withneb_cube_convolved.fits.gz'

circ_cube_off = SkyCube.read(circ_filename_off)

psf_table = psf_fromfits('irf_file.fits')
psfs = psf_table[3]

on_sizes = np.ones(int(config['binning']['enumbins'])) * u.deg
energarr = cube_on.energies('edges')
for idx in range(len(cube_on.energies('center'))):
    i = np.argmin(np.abs(energarr[idx].value - psf_table[0].value))
    j = np.argmin(np.abs(offset_fov - psf_table[2].value))
    on_sizes.value[idx] = psfs[0][j][i]
on_pos = SkyCoord(83.6333 * u.deg, 22.0144 * u.deg, frame='icrs')
off_pos = SkyCoord(83.6333 * u.deg, 22.0144 * u.deg, frame='icrs')
```

Spectral Analysis: General

2. Take counts spectrum:

```
on_region = CircleSkyRegion(on_pos, on_sizes[i])
off_region = CircleSkyRegion(off_pos, off_sizes[i])

on_data['value'][i] = cube_on.spectrum(on_region)['value'][i]
off_data['value'][i] = cube_off.spectrum(off_region)['value'][i]
```

3. Apply some criteria to define the fitting range:

```
if limasig >= 3 and on_data['value'][i] - off_data['value'][i] >= 7:

circ_on_vector = PHACountsSpectrum(energy_lo = cube_on.energies('edges')[:-1], energy_hi= cube_on.energies('edges')[1:],
data= circ_on_data['value'].data * circ_stats * u.ct, backscal = on_sizes[0].value, meta={'EXPOSURE' : livetime.value})

circ_off_vector = PHACountsSpectrum(energy_lo = circ_cube_off.energies('edges')[:-1], energy_hi=
circ_cube_off.energies('edges')[1:], data= circ_off_data['value'].data * circ_stats * u.ct, backscal = off_sizes[0].value,
meta={'EXPOSURE' : livetime.value, 'OFFSET' : 0.3 * u.deg})
```

Spectral Analysis: General

4. Make spectrum observation and fit

```
circ_sed_table = SpectrumObservation(on_vector = circ_on_vector, off_vector = circ_off_vector, aeff = aeff,
edisp = edisp)
```

```
fit_source = SpectrumFit(obs_list = ann_sed_table, model=models_ann_fit[k],forward_folded=True,
fit_range=(0.05 * u.Unit('TeV'), cube_on.energies('edges')[int(np.sum(ann_stats))]))
fit_source.fit()
fit_source.est_errors()
results = fit_source.result
```

5. Add systematical errors

   - Main source of errors

   - For now relaying in the current literature (MAGIC, HESS), supposing some factor of improvement (factor of 2 for example)

Spectral Analysis: General

- For now relaying in the current literature (MAGIC, HESS):

Systematical errors (%)

Parameter \ Model	LogParabola	ExponentialPowerLaw	PowerLaw
E_0	17	17	17
N_0	11	20	20
Index		4	2
Cutoff Energy		20	
α	2		
β	30		

Spectral Analysis: Nebula

- How much does the spectrum extends?

Time to detect the nebula at 5 sigma for the energy range displayed

MAGIC (Log-Parabola)		HESSII (Log-Parabola)	
Energy (TeV)	**$t(\sigma = 5)$ [h]**	**Energy (TeV)**	**$t(\sigma = 5)$ [h]**
$E < 0.1$	0.37	$E < 0.1$	3.8
$0.1 < E < 1$	0.0056	$0.1 < E < 1$	0.007
$E > 5$	0.17	$E > 5$	0.22
$E > 50$	30.3	$E > 50$	232.8

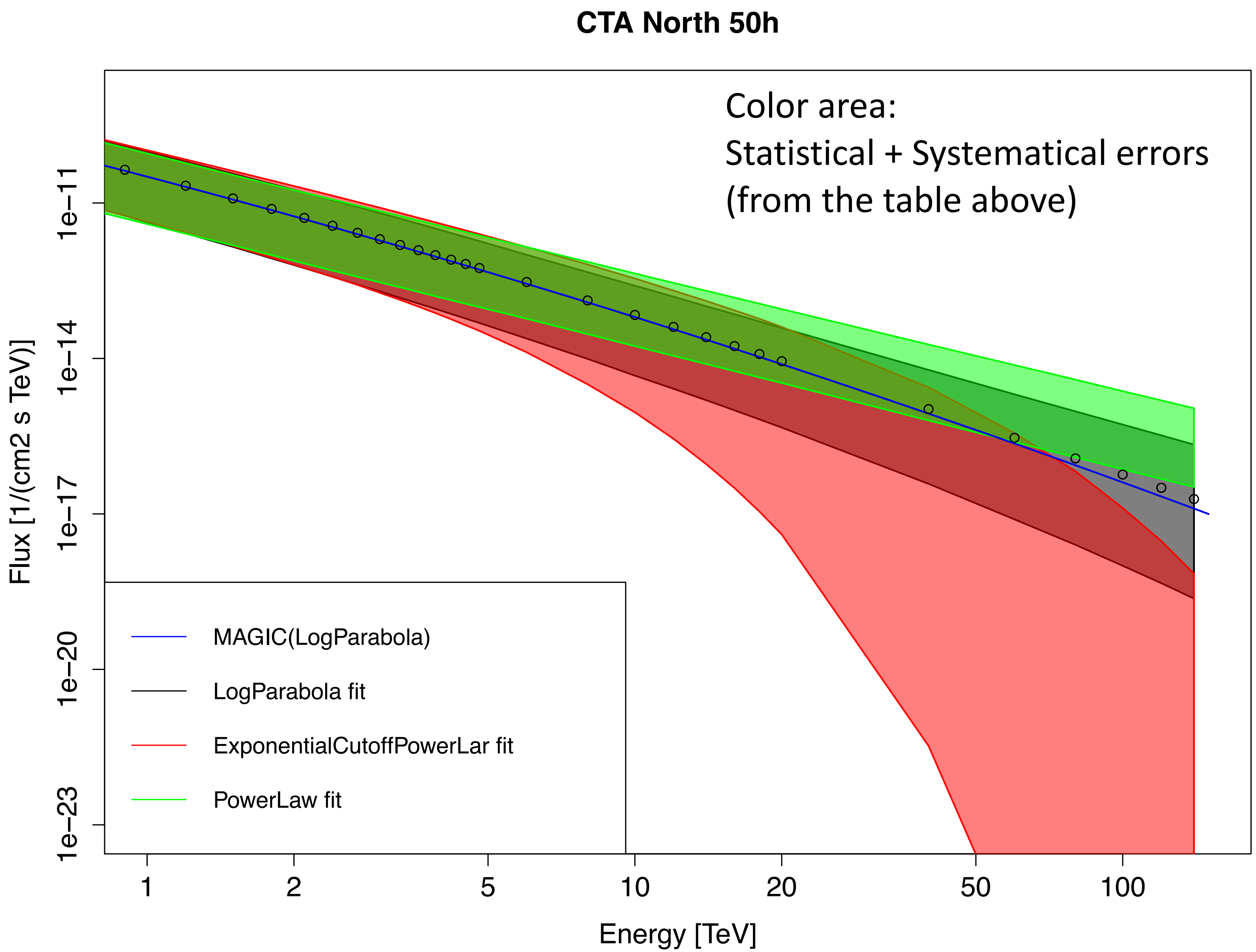
Ctools (30.3h -> $\sigma \sim$ 4.8)

Ctools (232.8h -> $\sigma \sim$ 4.5)

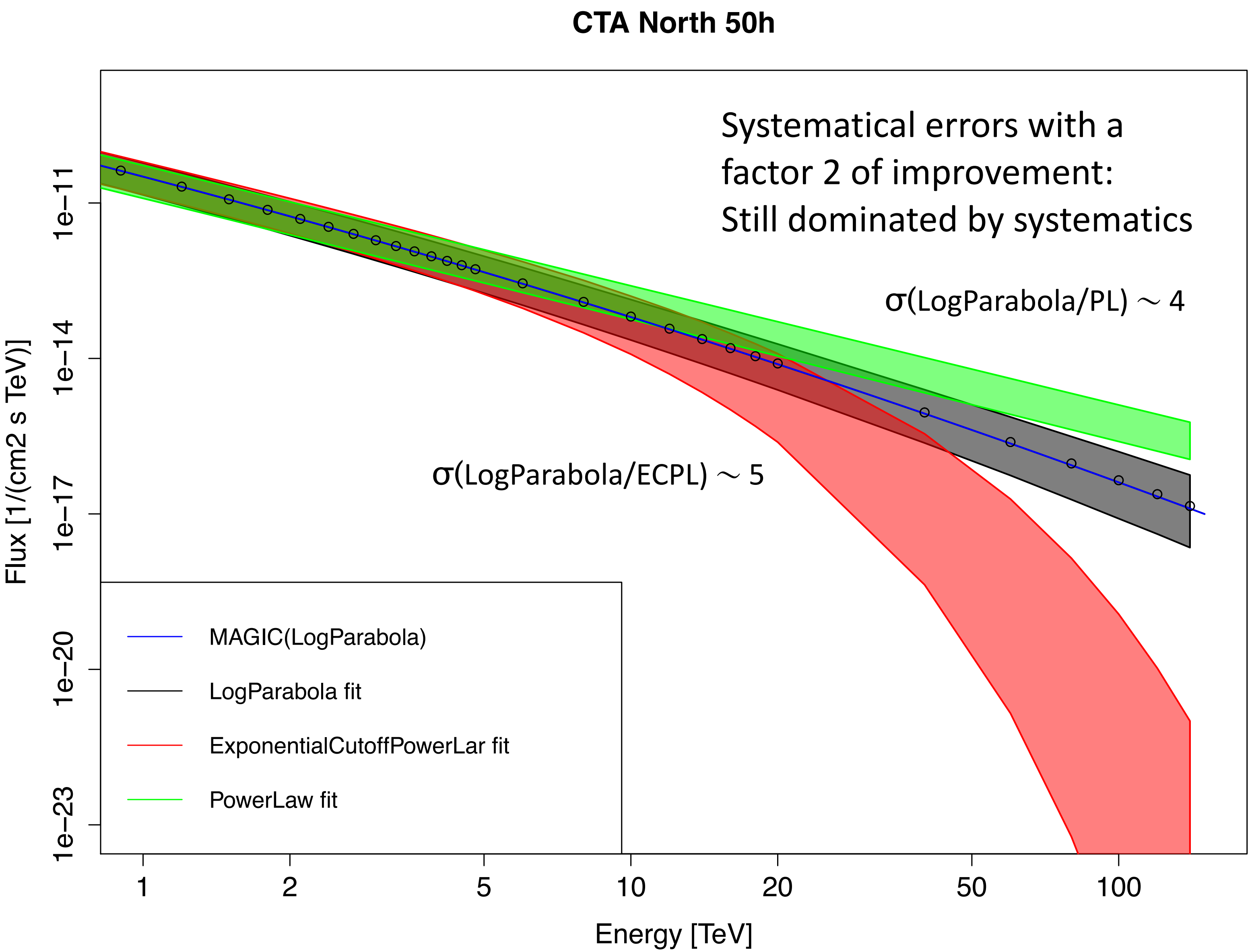
Spectral Analysis: Nebula

- How much does the spectrum extends? What is the spectral shape?

Spectral Analysis: Nebula



Spectral Analysis: Nebula



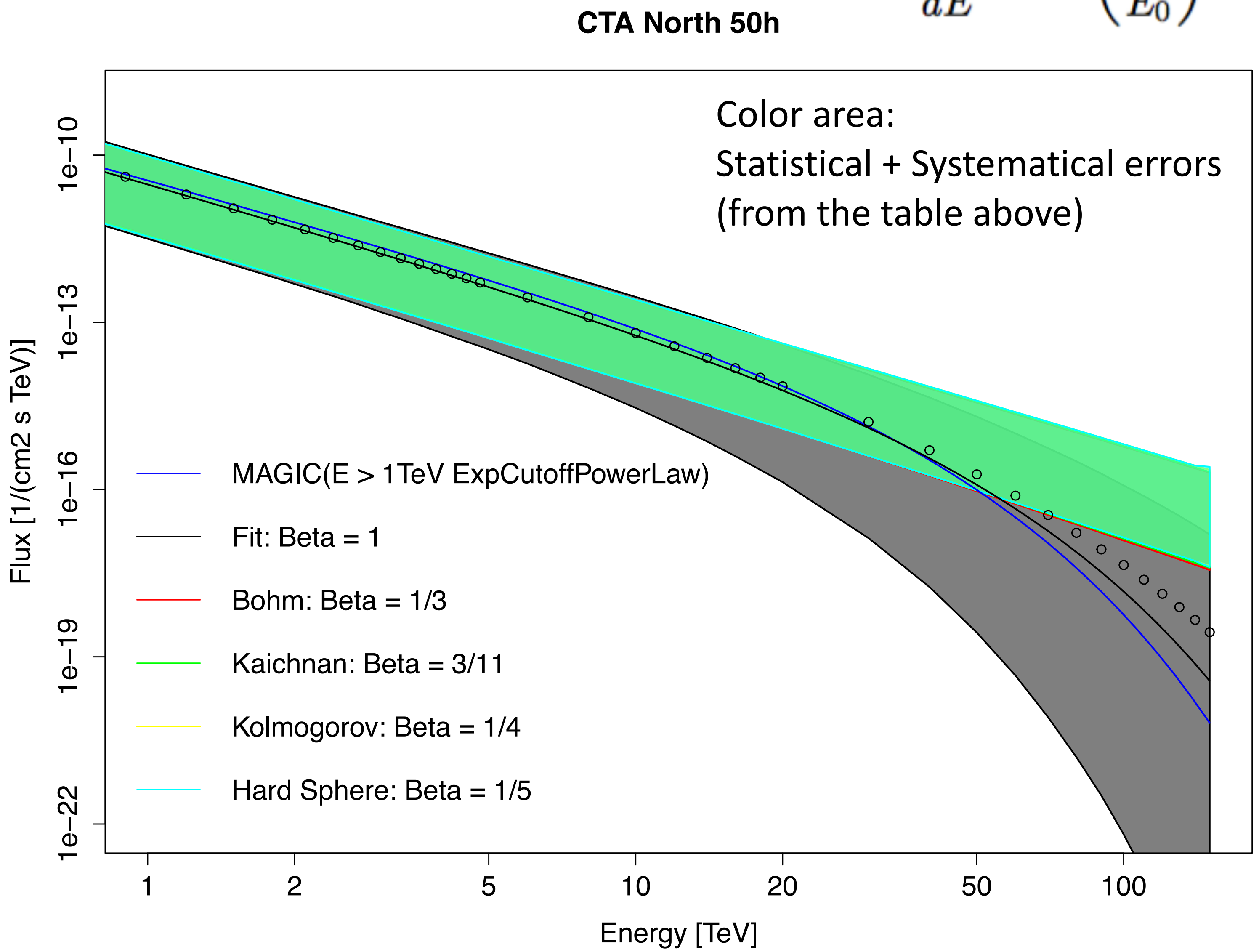
Spectral Analysis: Nebula

- How much does the spectrum extends? What is the spectral shape? How is the shape of the spectral cut-off?

$$\frac{dN}{dE} = N_0 \left(\frac{E}{E_0}\right)^{-\Gamma} \exp\left[-\left(\frac{E}{E_c}\right)^{\beta_\gamma}\right]$$

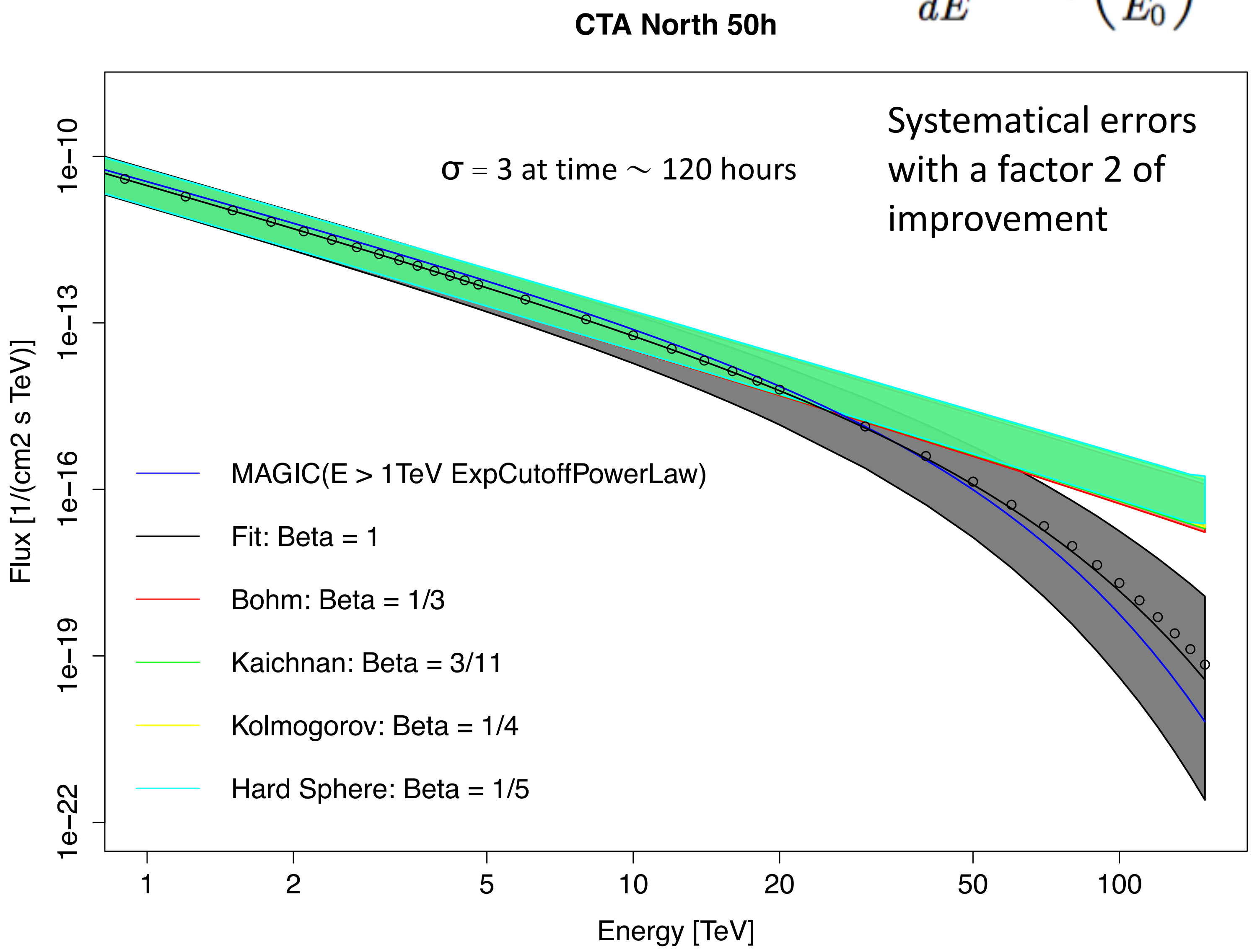
Spectral Analysis: Nebula

$$\frac{dN}{dE} = N_0 \left(\frac{E}{E_0}\right)^{-\Gamma} \exp\left[-\left(\frac{E}{E_c}\right)^{\beta_\gamma}\right]$$



Spectral Analysis: Nebula

$$\frac{dN}{dE} = N_0 \left(\frac{E}{E_0}\right)^{-\Gamma} \exp\left[-\left(\frac{E}{E_c}\right)^{\beta_\gamma}\right]$$



Spectral Analysis: Conclusions

- We needed at least a factor of two of improvement for the systematical errors in order to distinguish between the input and the other models.
- We can recover the initial spectral model with small statistical errors in comparison with systematics.
- We could distinguish sub-exponential cutoff models with differences in the β parameter greater than 0.67.

Morphology: General

1. Introduce the PSF

```
def psf_fromfits(filename):

    hdulist = pyfits.open(filename)
    hdu = hdulist[2]
    energy_lo = Quantity(hdu.data['ENERG_LO'][0], ‘TeV')
    energy_hi = Quantity(hdu.data['ENERG_HI'][0], ‘TeV')
    theta = Angle(hdu.data['THETA_LO'][0], 'deg')

    # Get sigmas
    shape = (len(theta), len(energy_hi))
    sigmas = []
    for key in ['SIGMA_1', 'SIGMA_2', 'SIGMA_3']:
        sigma = hdu.data[key].reshape(shape).copy()
        sigmas.append(sigma)

    # Get amplitudes
    norms = []
    for key in ['SCALE', 'AMPL_2', 'AMPL_3']:
        norm = hdu.data[key].reshape(shape).copy()
        norms.append(norm)

    return dict(energy_lo = energy_lo, energy_hi = energy_hi, theta = theta, sigmas = sigma, norms = 
norms)
psf_table = psf_fromfits('irf_file.fits')
```

For the moment we just use a simple gaussian with σ_1

Morphology: General

2. Fit morphology for each energy bin I

```
filename = 'nexcess_cube_convolved.fits.gz'         ## Read file to fit
cube = SkyCube.read(filename)

config = read_config('config.yaml')
binsz = config['binning']['binsz']
offset_fov = config['selection']['offset_fov']

# Take PSF data
irffile = 'irf_file.fits'
psf_table = psf_fromfits(irffile)

energarr = cube.energies('edges')
sigmas = psf_table[3]
norms = psf_table[4]

hdu = pyfits.open(filename)
im_sizex = hdu[0].header['NAXIS1']
im_sizey = hdu[0].header['NAXIS2']
cx = 0.5*im_sizex
cy = 0.5*im_sizey

# Check the significance…
# Make image cube from slice excess convolved cube
cube_sum = np.zeros((cube.data.shape[1],cube.data.shape[2])) * u.ct
cube_sum = np.add(cube_sum, cube.data[idx])
image_sum = SkyCube.empty_like(cube)
image_sum.data = cube_sum

image_sum.write('sum_image.fits.gz', overwrite=True)
# Find nearest energy and theta value
i = np.argmin(np.abs(energarr[idx].value - psf_table[0].value))
j = np.argmin(np.abs(offset_fov - psf_table[2].value))

# Make PSF
s1 = sigmas[0][j][i]/binsz
s2 = sigmas[1][j][i]/binsz
s3 = sigmas[2][j][i]/binsz
ampl = norms[0][j][i]
ampl2 = norms[1][j][i]
ampl3 = norms[2][j][i]
```

Morphology: General

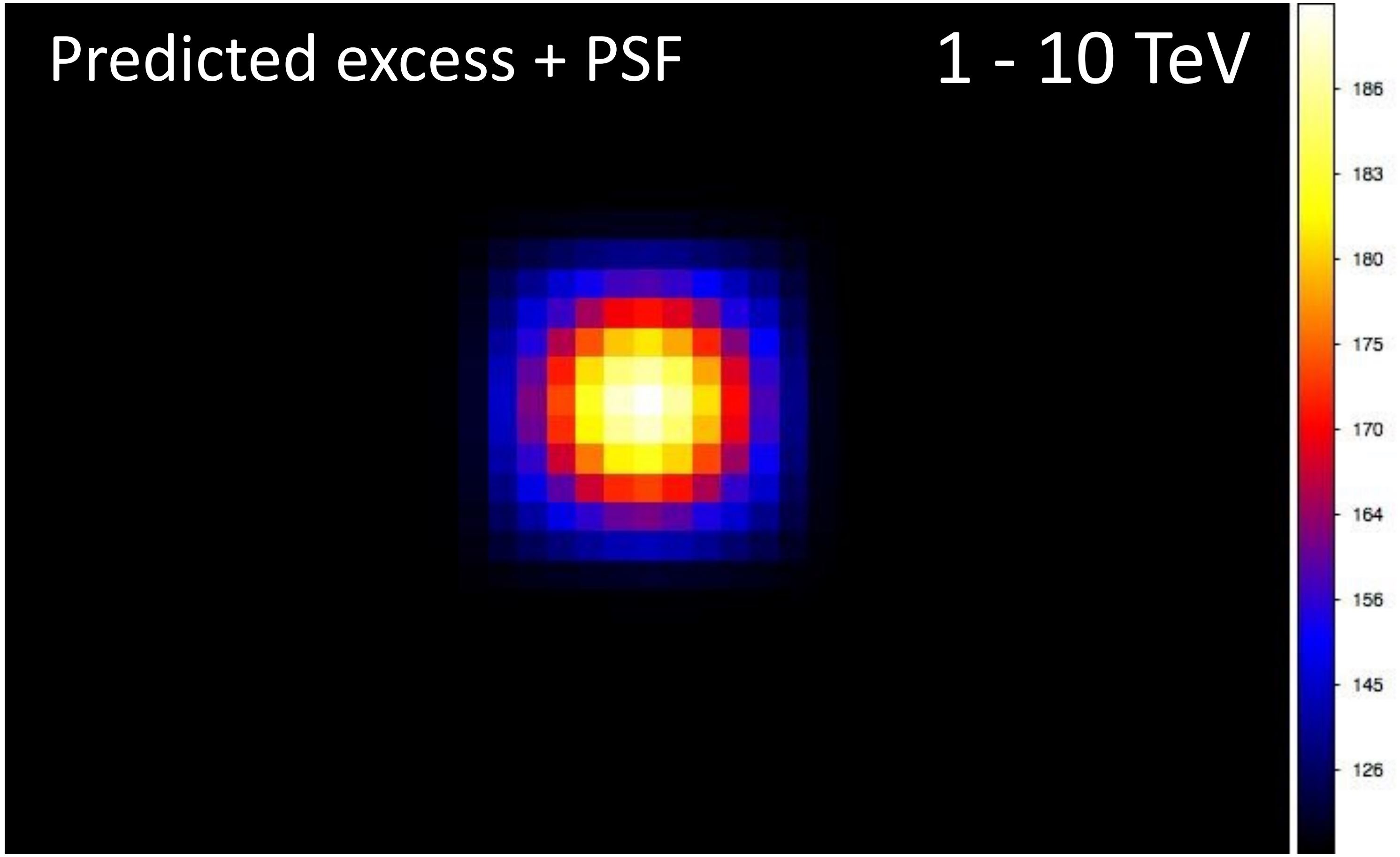
3. Fit morphology for each energy bin II. We use sherpa

```
# Morphological fitting
load_image(“sum_image.fits.gz")

set_method("simplex")
set_stat(“cash")

x0 = 125
y0 = 125
rad0 = 48.0

image_getregion(coord="physical")
'circle(x0,y0,rad0);'

notice2d("circle(" + str(x0) + "," + str(y0) + "," + str(rad0) + “)")

load_user_model(GaussianSource, “sph2d”)
add_user_pars( "sph2d", ["sigma1", "sigma2", "sigma3", "alpha", "beta", "ampl", "size", "xpos", "ypos"] )
set_model(sph2d + const2d.bgnd)

show_model()
fit()
conf()


save_model("model_" + str(idx) + ".fits")
save_resid("resid_" + str(idx) + ".fits")
```

Morphology: General

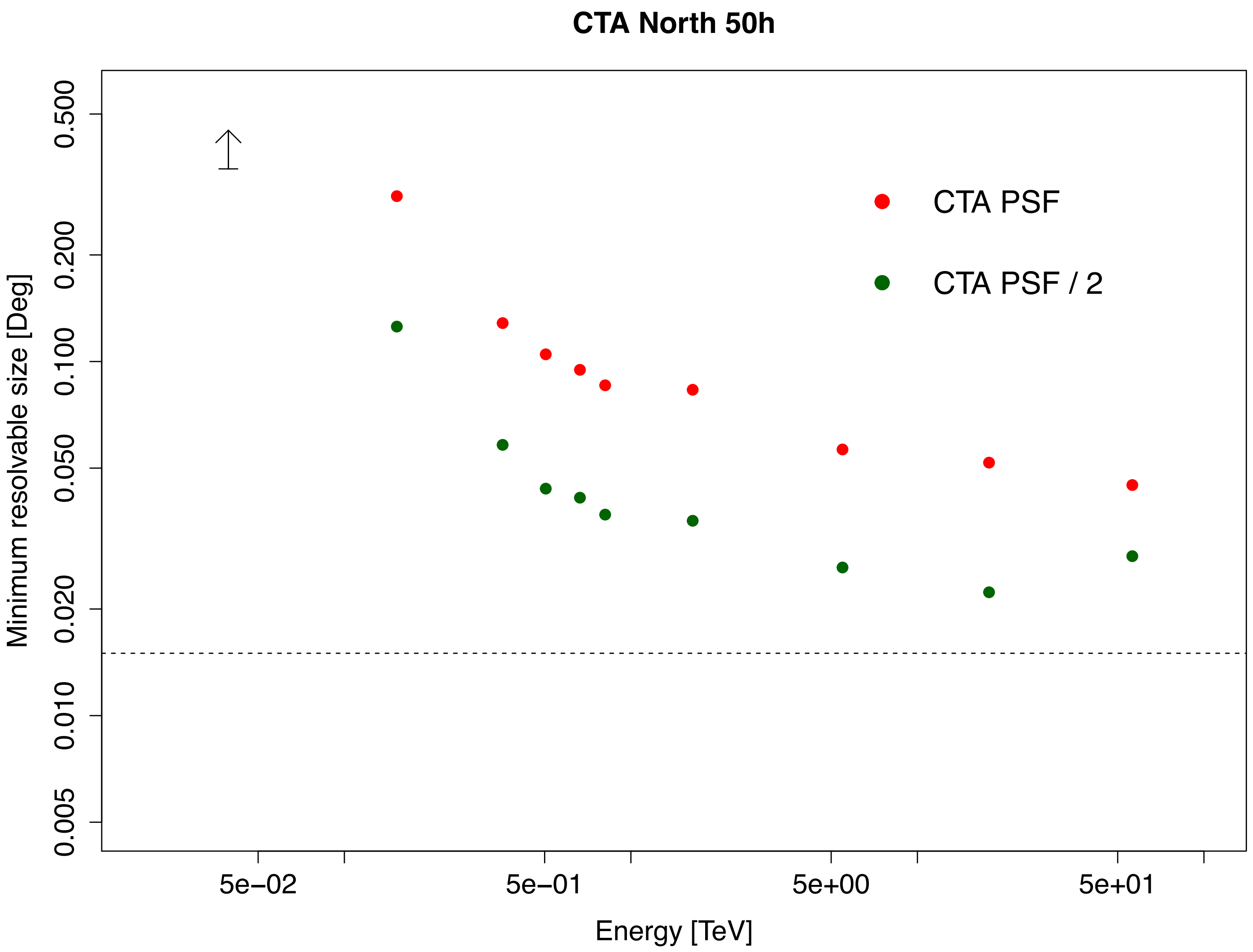
- Example of cube. Excess convolved with PSF :

- Model:
  - MAGIC Log-Parabola
  - 2D sphere 52''
  - Pulsar index: 2.9
  - Point-like pulsar
  - Time of 30 hours
  - PSF ~ 2’
- **Question: To recover the TeV source extension (Sherpa)**



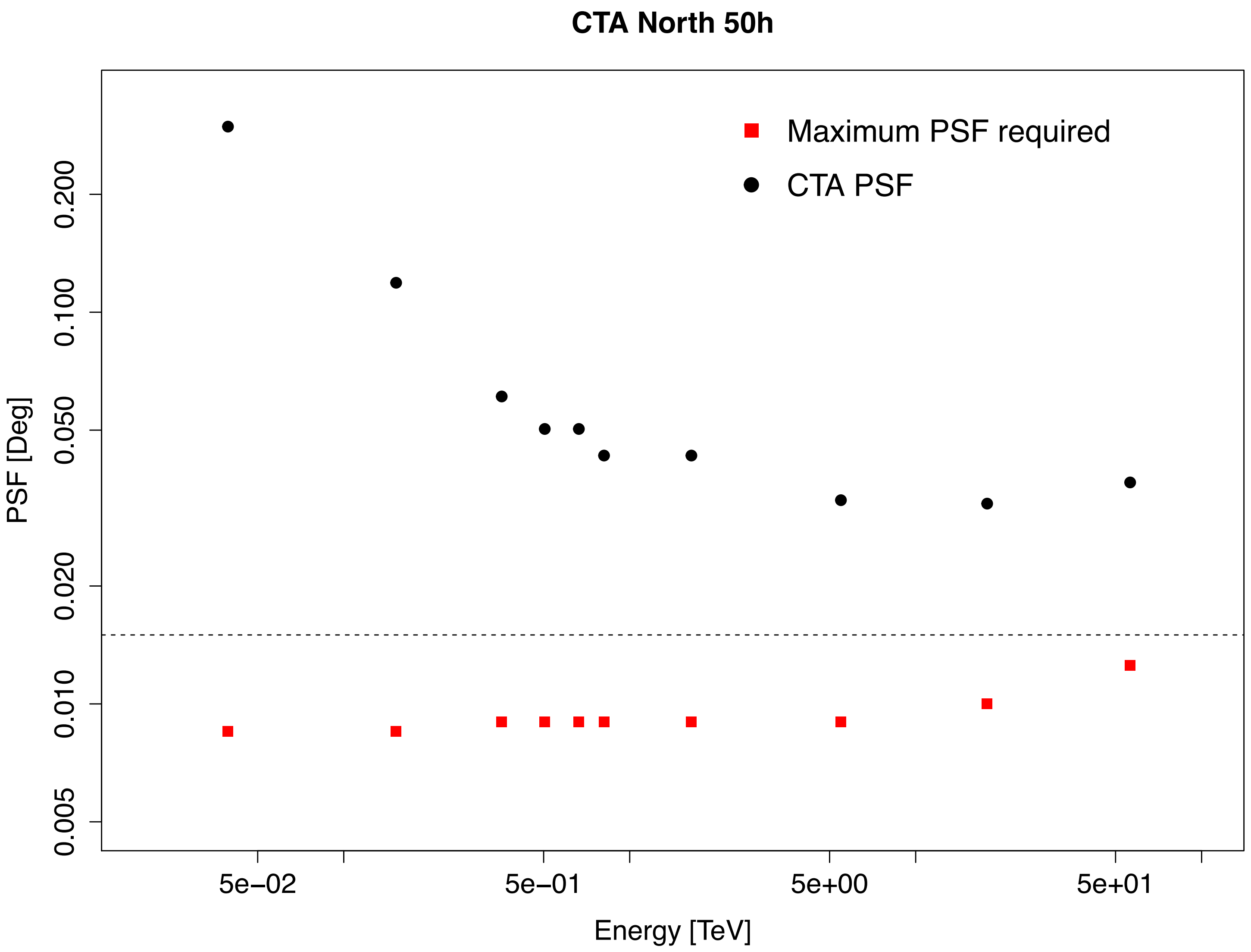
Morphology: General

- We can either fix the CTA IRFs and change the size of the source until the one is resolved:
- Model:
  - MAGIC Log-Parabola
  - 2D sphere 52'' (dashed line)
  - Pulsar index: 2.9
  - Point-like pulsar
  - Time of 50 hours



Morphology: General

- Or fix the source size and change the PSF convolved in the cube generator until it is resolved:
- Model:
  - MAGIC Log-Parabola
  - 2D sphere 52'' (dashed line)
  - Pulsar index: 2.9
  - Point-like pulsar
  - Time of 50 hours



To conclude:

- The cube generation allow us to place simulations of two objects in the same position with different morphology and spectral characteristics plus the background.
- From the simulated cubes it is possible to make spectral and morphological analysis with gammapy and sherpa.
- We can apply this tools to look for what to expect of the Crab Nebula and Pulsar regarding to; detection significance, spectral model or morphology with the CTA observatory