EVENT SAMPLING FOR GAMMAPY

Fabio Pintore
Axel Donath
Andrea Giuliani

Gammapy Coding Sprint, February 25th, 2019

Aim of the project

Develop an event sampler, basis of an event list simulator.

- *Requirement for the CTA Science Tools*
- *CTA DC2*
- *Useful for general tests*

Open issue #761 (https://github.com/gammapy/gammapy/issues/761)

From a given spatial and spectral distribution, and exposure time, the simulator should:

- *simulate a number of events with true energy and (ra,dec) position*
- *apply IRF (Edisp and PSF) to each event*
- *simulate the background*
- *provide an output event list*

Main steps of the simulator:

- Compute the cumulative of the total predicted events (npred = aeff x obs_time x flux)

- Sample random events from the total number of events

- Sample true energies and positions from the npred distribution

- For each energy, sample the Edisp and get the reconstructed energy

- *Sample the 1-dim radially symmetric PSF to get RA_reco, DEC_reco*

- *include the light-curve*

New proposed classes: InverseCDFSampler

```python
class InverseCDFSampler:
    """Inverse CDF sampler

    Parameters
    ----------
    pdf : ...


    """
    def __init__(self, pdf, axis=None, random_state=0):
        self.random_state = get_random_state(random_state) #determines a set of random numbers
        self.axis = axis

        if axis is not None:
            self.cdf = np.cumsum(pdf, axis=self.axis)
            self.cdf /= self.cdf[:, [-1]]
        else:
            self.pdf_shape = pdf.shape  #gives the shape of the PDF array

            pdf = pdf.ravel() / pdf.sum()  #flattens the array along one axis
            self.sortindex = np.argsort(pdf, axis=None) #sorting of the elements and giving the indexes

            self.pdf = pdf[self.sortindex]  #sort the pdf array
            self.cdf = np.cumsum(self.pdf)  #evaluate the cumulative sum of the PDF array

    def sample_axis(self):
        """Sample along a given axis.
        """
        choice = self.random_state.uniform(high=1, size=len(self.cdf))

        #find the indices corresponding to this point on the CDF
        index = np.argmin(np.abs(choice.reshape(-1, 1) - self.cdf), axis=self.axis)

        return index + self.random_state.uniform(low=-0.5, high=0.5,
                                                 size=len(self.cdf))
```

New proposed classes: InverseCDFSampler

```
    def sample(self, size):
        """Draw sample from the given PDF.

        Parameters
        ----------
        size : int
            Number of samples to draw.

        Returns
        -------
        index : tuple of `~numpy.ndarray`
            Coordinates of the drawn sample
        """
        #pick numbers which are uniformly random over the cumulative distribution function
        choice = self.random_state.uniform(high=1, size=size)

        #find the indices corresponding to this point on the CDF
        index = np.searchsorted(self.cdf, choice)
        index = self.sortindex[index]

        # map back to multi-dimensional indexing
        index = np.unravel_index(index, self.pdf_shape)
        index = np.vstack(index)

        index = index + self.random_state.uniform(low=-0.5, high=0.5,
                                                  size=index.shape)
        return index
```

New proposed classes: MapEventSampler

```
class MapEventSampler:
    """Map event sampler

    Parameters
    ----------

    npred_map : `~gammapy.maps.Map`
        Predicted number of counts map.
    ...


    """
    def __init__(self, npred_map, psf_map=None, edisp_map=None, background_map=None,
                 random_state=0):
        self.random_state = get_random_state(random_state)
        self.npred_map = npred_map
        self.psf_map = psf_map
        self.edisp_map = edisp_map
        self.background_map = background_map

    def npred_total(self):
        #return self.random_state.poisson(self.npred_map.data.sum())
        return self.random_state.poisson(np.sum(self.npred_map.data))

    def apply_edisp(self, events):
        # apply energy dispersion to list of events
        pdf = self.edisp_map.interp_by_coord({"skycoord": events.radec,
        "energy": events.energy})

        return events

    def apply_psf(self, events):
        # apply psf to list of events
        rad = np.linspace(0, 1 * u.deg, 100)[np.newaxis, :]
        pdf = self.psf_map.interp_by_coord({"skycoord": events.radec,
        "energy": events.energy, "rad": rad})

        # sample from pdf along rad axis
        return events
```

New proposed classes: MapEventSampler

```python
def sample_background(self):
    self.background_map
    # sample from background model without applying IRFs
    return events
```

```python
def sample_npred(self):
    n_events = self.npred_total()

    cdf_sampler = InverseCDFSampler(self.npred_map.data, random_state=self.random_state)

    pix_coords = cdf_sampler.sample(n_events)
    coords = self.npred_map.geom.pix_to_coord(pix_coords[::-1])

    events = Table()
    events['lon_true'] = coords[0] * u.deg
    events['lat_true'] = coords[1] * u.deg
    events['e_true'] = coords[2] * u.TeV
    return events
```

```python
def sample_events(self):
    """Sample all events.

    Returns
    -------

    """
    events = self.sample_npred()
    events = self.apply_psf(events)
    events = self.apply_edisp(events)

    bkg_events = self.sample_background()

    table = vstack(events, bkg_events)

    return table
```

Preliminary event sampler

```
spatial_model = SkyGaussian("0 deg", "0 deg", sigma="0.2 deg")
spectral_model = PowerLaw(amplitude="1e-11 cm-2 s-1 TeV-1")
skymodel = SkyModel(spatial_model=spatial_model, spectral_model=spectral_model)
```

```
position = SkyCoord(0.0, 0.0, frame='galactic', unit='deg')

energy_axis = MapAxis.from_bounds(1, 100, nbin=30, unit="TeV", name="energy", interp="log")

exposure = Map.create(
    binsz=0.02,
    map_type='wcs',
    skydir=position,
    width="5 deg",
    axes=[energy_axis],
    coordsys="GAL", unit="cm2 s"
)

exposure.data = 1e13 * np.ones(exposure.data.shape)
```

```
evaluator = MapEvaluator(model=skymodel, exposure=exposure)

npred = evaluator.compute_npred()
```

```
sampler = MapEventSampler(npred)
events = sampler.sample_npred()
```

Preliminary event sampler

lon_true	lat_true	e_true
deg	deg	TeV
float64	float64	float64
359.7333370969594	-0.07244496321415028	1.5217603637521027
359.85837797220023	0.20359207355119283	2.1419659842005205
359.8918551765717	-0.019506425432173274	5.622337036174219
359.7953553171566	-0.1186550073707349	1.7314521299420373
0.15735024730484043	0.2935878554699713	1.7356792123034828
359.9989314563643	-0.03092606310887902	1.7304144260407368
0.005488112715788418	-0.06926841577782568	1.206531807265262
359.7997714508275	0.17793342586080713	3.3898717477327427
359.8245883850299	-0.08019322105206583	1.4559340127482747
...	...	...
359.9948263186622	-0.0035576453361150584	30.135578560595683
359.98301923383144	-0.23620304176194482	1.5714171647061466
359.93933390746906	-0.24977362034907089	5.199601411815321
0.10082034556273044	0.06448634057949448	17.08823681133315
0.012892623030561481	0.08195688968988066	3.5358538730155957
359.65286586219196	0.11724383034843357	41.39587804204178
0.17967342994632587	-0.10054161021953746	1.1763465343878494
0.1362953534952763	0.4392166931612599	78.63715177583421
359.7319748099839	-0.021868890015576313	1.2230749144976887
0.0014141716539458572	0.25548094665397286	6.022214671662589

Preliminary event sampler

```
filename = "$GAMMAPY_DATA/cta-1dc/caldb/data/cta/1dc/bcf/South_z20_50h/irf_file.fits"
psf_gauss = EnergyDependentMultiGaussPSF.read(filename=filename, hdu="POINT SPREAD FUNCTION")
psf_3d = psf_gauss.to_psf3d(rad=np.linspace(0, 1, 100) * u.deg)
```

```
theta_axis = MapAxis.from_bounds(0, 0.5, nbin=100, unit="deg", name="theta")

geom_psf = WcsGeom.create(
    binsz=0.2,
    skydir=position,
    width="5 deg",
    axes=[theta_axis, energy_axis],
    coordsys="GAL"
)

psf_map = make_psf_map(psf_3d, geom=geom_psf, pointing=geom_psf.center_skydir, max_offset=3 * u.deg)
```

```
events.sort("e_true")
coord = {
    "lon": events["lon_true"].reshape(-1, 1),
    "lat": events["lat_true"].reshape(-1, 1),
    "energy": events["e_true"].quantity.reshape(-1, 1),
    "theta": (theta_axis.center * theta_axis.unit)
}

pdf = psf_map.psf_map.interp_by_coord(coord)
```

TODO List for the simulator prototype:

- add sampling of phi angle for PDF
- compute the reconstructed positions of the events

- add application of energy dispersion
- compute reconstructed energy of the events

- sampling of the background events

- Include light-curves simulation

We started to write a new PIG (#009)

```
.. _pig-009:

************************
PIG 9 - Event simulator
************************

* Author: Fabio Pintore, Andrea Giuliani, Axel Donath
* Created: Feb 04, 2019
* Accepted:
* Status:
* Discussion:

Abstract
========

- create an event simulator
- required by the CTA Science Tools
- necessary to simulate the DC2


Proposal
========

- the design follows the ASTRIsim simulator
- the latter requests the expected source flux cube for a given spectrum and morphology, prior the application of the IRF
- (to the counts-cube can be associated a light-curve)
- expected background counts cube evaluated from the IRF
- the energy dispersion is applied to the source counts
- the PSF is applied to the source counts
- source and background events are stacked together
- write them all into an event list file (.fits)

We would like to improve the class presented in general_random_array.py issue. We propose to add the sorting of the cumulative distribution
and then we also want to provide an interpolation along the bins. This class has to be extended sampling along a given axis and not along all
the array.

.. code::

    class InverseCDFSampler:
         """Inverse CDF folder"""
         def __init__(self, pdf, random_state=0):
             self.random_state = get_random_state(random_state)
             self.pdf_shape = pdf.shape

             pdf = pdf.ravel() / pdf.sum()
             self.sortindex = np.argsort(pdf, axis=None)

             self.pdf = pdf[self.sortindex]
             self.cdf = np.cumsum(self.pdf)
```