Low level analysis code

The case of image and cube analysis

What we have now

- Currently image and cube processing rely on `SkyImage` and `SkyCube` classes

- Image & Cube processing performed by high level classes which perform all the analysis sequentially i.e. observation per observation (exposure, background, count maps, etc):

    - `SingleObsImageMaker & StackedObsImageMaker`

    - `IACTBasicImageEstimator`

    - `SingleObsCubeMaker & StackedObsCubeMaker`

- This approach is not modular and creates a lot of code duplication.

- Some cube-related analysis is required for images creating some cross-dependencies.

Low level analysis for images & cubes

- The low-level analysis cube package deals with the production of all maps/cubes and PSF kernels required to perform 2D and 3D modeling and fitting. This includes counts, exposure, hadron acceptance and normalized background maps and cubes.

- The building blocks on which it relies are coded in:

  - `gammapy.data`

    - `EventList`, `DataStore`, `DataStoreObservation`

  - `gammapy.maps`

    - `WcsNDMap` used both for images and cubes

  - `gammapy.irf`

    - `EffectiveAreaTable2D`, `EnergyDispersion2D`, `EnergyDependentTablePSF`, etc

Low level analysis for images & cubes

- The low level analysis is performed on an observation per observation (or GTI) basis.

- This is required by the response and background rapid variations. Therefore, all basic functions operate on a single `EventList` or set of IRFs (i.e. `EffectiveAreaTable2D`, `EnergyDispersion2D`, `EnergyDependentTablePSF`).

- The iterative production of the individual reduced datasets and IRFs and their combination is realized by the higher level class.

- The individual observation products can be serialized, mostly for analysis debugging purposes or to avoid reprocessing large databases when new data are added.

- Extract cutouts geometry for a given position and size
- Use astropy Cutout2D
- Use/Store slices?
- Can use `WcsNDMap.crop ?`

```
def make_cutout(ndmap, position, size, margin = 0.1*u.deg):
    """Create a cutout of a WcsNDMap around a given direction.

    Parameters
    ----------
    ndmap : `~gammapy.maps.WcsNDMap`
            the map on which the cutout has to be extracted
    position : `~astropy.coordinates.SkyCoord`
            the center position of the cutout box
    size : Tuple of `~astropy.coordinates.Angle`
            the angular sizes of the box
    margin : `~astropy.coordinates.Angle`
            additional safety margin

    Returns
    -------
    cutout : `~gammapy.maps.WcsNDMap`
             the cutout map itself
    cutout_slice :

    """
```

```
def make_map_counts(evts, ref_geom, pointing, offset_max):
    """ Build a WcsNDMap (space - energy) with events from an EventList.
    The energy of the events is used for the non-spatial axis.

    Parameters
    ----------
    evts : `~gammapy.data.EventList`
           the input event list
    ref_geom : `~gammapy.maps.WcsGeom`
        Reference WcsGeom object used to define geometry (space - energy)
    offset_max : `~astropy.coordinates.Angle`
        Maximum field of view offset.

    Returns
    -------
    cntmap : `~gammapy.maps.WcsNDMap`
        Count cube (3D) in true energy bins
    """
```

- counts maps with given WcsGeom
- Use maximum offset. Could use valid mask instead.

- Exposure in true energy (i.e. non convolved by EDISP)
- For 3D analysis and fine binning

```
def make_map_exposure_true_energy(pointing, livetime, aeff, ref_geom, offset_max):
    """Compute exposure WcsNDMap in true energy (i.e. not convolved by Edisp).

    Parameters
    ----------
    pointing : `~astropy.coordinates.SkyCoord`
        Pointing direction
    livetime : `~astropy.units.Quantity`
        Livetime
    aeff : `~gammapy.irf.EffectiveAreaTable2D`
        Effective area table
    ref_geom : `~gammapy.maps.WcsGeom`
        Reference WcsGeom object used to define geometry (space - energy)
    offset_max : `~astropy.coordinates.Angle`
        Maximum field of view offset.

 rns
 ---
 ap : `~gammapy.maps.WcsNDMap`
 Exposure cube (3D) in true energy bins
```

```
def make_map_exposure_reco_energy(pointing, livetime, aeff, edisp, spectrum, ref_geom, offset_max):
    """ Compute exposure WcsNDMap in reco energy (i.e. after convolution by Edisp and assuming a true
    energy spectrum).
    This is useful to perform 2D imaging studies.

    Parameters
    ----------
    pointing : `~astropy.coordinates.SkyCoord`
        Pointing direction
    livetime : `~astropy.units.Quantity`
        Livetime
    aeff : `~gammapy.irf.EffectiveAreaTable2D`
        Effective area table
    edisp : `~gammapy.irf.EnergyDispersion2D`
        Energy dispersion table
    spectrum : `~gammapy.spectrum.models`
        Spectral model
    ref_geom : `~gammapy.maps.WcsGeom`
        Reference WcsGeom object used to define geometry (space - energy)
    offset_max : `~astropy.coordinates.Angle`
        Maximum field of view offset.
    etrue_bins : `~astropy.units.Quantity`
        True energy bins (edges or centers?)

    Returns
    -------
    expmap : `~gammapy.maps.WcsNDMap`
        Exposure cube (3D) in reco energy bins
    """
```

- Exposure in reco energy
- Assumes a spectrum and convolves with EDISP
- For 2D analysis
- `etrue_bins` input for energy integration

- hadron acceptance map i.e. predicted background counts
- This requires integration of bkg3D IRF
- Will usually require proper normalization before fitting

```python
def make_map_hadron_acceptance(pointing, livetime, bkg, ref_geom, offset_max):
    """Compute hadron acceptance cube i.e.  background predicted counts.

    This function evaluates the background rate model on
    a WcsNDMap, and then multiplies with the cube bin size,
    computed via ???, resulting
    in a cube with values that contain predicted background
    counts per bin.
    The output cube is - obviously - in reco energy.

    Note:
    -----
    bkg.evaluate should be replaced with a function returning directly an integrated bkg flux.

    Parameters
    ----------
    pointing : `~astropy.coordinates.SkyCoord`
        Pointing direction
    livetime : `~astropy.units.Quantity`
        Observation livetime
    bkg : `~gammapy.irf.Background3D`
        Background rate model
    ref_cube : `~gammapy.maps.WcsGeom`
        Reference cube used to define geometry
    offset_max : `~astropy.coordinates.Angle`
        Maximum field of view offset.

    Returns
    -------
    background : `~gammapy.maps.WcsNDMap`
        Background predicted counts sky cube in reco energy
    """
```

```python
def make_map_FoV_background(acceptance_map, counts_map, excluded_map):
    """ Build Normalized background map from a given acceptance map and count map.
    This operation is normally performed on single observation maps.
    An exclusion map is used to avoid using regions with significant gamma-ray emission.
    All maps are assumed to follow the same WcsGeom.

    Note
    ----
    A model map could be used instead of an exclusion mask.

    Parameters
    ----------
    acceptance_map : `~gammapy.maps.WcsNDMap`
        the observation hadron acceptance map (i.e. predicted background map)
    counts_map : `~gammapy.maps.WcsNDMap`
        the observation counts map
    excluded_map : `~gammapy.maps.WcsNDMap`
        the exclusion mask

    Return
    ------
    norm_bkg_map :: `~gammapy.maps.WcsNDMap`
        the normalized background
    """
```

- Normalize background map using all events in FoV
- Requires an energy grouping scheme

```python
def make_map_ring_background(ring_estimator, acceptance_map, counts_map, excluded_map):
    """ Build normalized background map from a given acceptance map and count map using
    the ring background technique.
    This operation is performed on single observation maps.
    An exclusion map is used to avoid using regions with significant gamma-ray emission.
    All maps are assumed to follow the same WcsGeom.

    Note that the RingBackgroundEstimator class has to be adapted to support WcsNDMaps.

    Parameters
    ----------
    ring_estimator: `~gammapy.background.AdaptiveRingBackgroundEstimator` or `RingBackgroundEstimator`
         the ring background estimator object
    acceptance_map : `~gammapy.maps.WcsNDMap`
         the observation hadron acceptance map (i.e. predicted background map)
    counts_map : `~gammapy.maps.WcsNDMap`
         the observation counts map
    excluded_map : `~gammapy.maps.WcsNDMap`
         the exclusion mask

    Return
    ------
    norm_bkg_map :: `~gammapy.maps.WcsNDMap`
         the normalized background
    """
```

- Ring background estimator
- return normalized background
- Estimator object contains OFF map, exposure ON and OFF maps

- Global reduction performed by a general class/script

```python
## This is just a basic example of what such a class could look like
class MakeMaps(object):
    """ Make all basic maps for a single Observation.

    Parameters
    ----------
    ref_geom : `~gammapy.maps.WcsGeom`
           the reference image geometry
    offset_max : `~astropy.coordinates.Angle`
           maximum offset considered
    """
```

```python
def __call__(self,obs, write=None):

    # First make cutout of the global image
    try:
        excluded_map_cutout, cutout_slices = make_cutout(self.exclusion_map, obs.pointing_radec ,
                                                         [2*self.offset_max, 2*self.offset_max])
    except PartialOverlapError:
        print("Observation {} not fully contained in target image. Skipping it.".format(obs.obs_id))
        return

    cutout_geom = excluded_map_cutout.geom

    # Make count map
    count_obs_map = make_map_counts(obs.events, cutout_geom, obs.pointing_radec, self.offset_max)

    # Make exposure map
    expo_obs_map = make_map_exposure_true_energy( obs.pointing_radec, obs.observation_live_time_duration,
                                                  obs.aeff, cutout_geom, self.offset_max)

    # Make hadron acceptance map
    acceptance_obs_map = make_map_hadron_acceptance( obs.pointing_radec, obs.observation_live_time_duration,
                                                     obs.bkg, cutout_geom, self.offset_max)

    # Make normalized background map
    background_obs_map = make_map_FoV_background(acceptance_obs_map, count_obs_map, excluded_map_cutout)

    self._add_cutouts(cutout_slices, count_obs_map, expo_obs_map, background_obs_map)
```

Many actions…

- Add metadata (`OrderedDict`) and units to maps (esp. serialization)
- Improve Cutout/Slices approach
- Modify background IRF (`background3D`).
  Define `.integrate` method. Same for `EffectiveArea2D?`
- Implement/test bkg normalization scheme
- Work on exposure maps
- Develop PSFKernel, EDISPKernel classes for convolution