PSF GENERATION WITH ALIKE

ANDREW CHEN AND TOMASO CONTESSI

1. Areas of the annuli

The area of each annulus as a function of angle ρ from the true position is: $d\Omega = 2\pi sin(\rho)d\rho$.

2. PSF Weighting as a function of energy

For each true energy E_i from 0 to ∞: $WS_i(E_i) = E_i^{1-\alpha} - E_{i+1}^{1-\alpha}$.
2) Define the energy dispersion weight:

$$WE_i(E_i, \theta, 0) = \sum_{E_j=E_{min}}^{E_{max}} EDP(E_i, E_j, \theta, 0)$$

where E_j are the reconstructed energies.

3. Total PSF

In practice the final PSF is:

$$PSF(\rho) = \frac{\sum\limits_{E_i=0}^{\infty} PSF(\rho, 0, \theta, 0, E_i) \times A_{eff}(E_i, \theta, 0) \times WE_i(E_i, \theta, 0) \times WS_i(E_i)}{\sum\limits_{E_i=0}^{\infty} A_{eff}(E_i, \theta, 0) \times WE_i \times WS_i}.$$

In reality, instead of calculating the denominator, the program renormalizes the numerator: $\sum\limits_{\rho=0}^{\rho_{max}} PSF(\rho) \times d\Omega = 1$.

4. Description of the function AlikePSFMap::CalcWeighted

Vector PSF is evaluated by the following function:

```
TVectorD AlikePsfArray::CalcWeighted(
     Float_t   index,   // IN: Energy spectral index
     Float_t   E\_low,  // IN: Lowest considered energy (application option)
     Float_t   E\_high, // IN: Highest considered energy (application option)
     Float_t   theta,   // IN: angle from instrument axis
     TVectorD* rho);    // OUT: rho values as from PFS file
```

Other input data used by this function are:
- rhoArr: list of angles where rhoArr(i+1)-rhoArr(i) = 2*rhoArr(0) = $\Delta\varrho$
- psfenergies: psf energies channels for each rho, from PSF file
- other values from PSF file (see step 5)
- values from AEFF file (see step 5)
- values from EDP file (see step 4)

The function returns in rho output parameter a copy of rhoArr, and the weighted PSF array as the function return value (psfArr).

The function performs the following steps:

4.1. If index is negative its sign is changed

4.2. Evaluation of lowetrue and highetrue as the indexes that make the interval [psfenergies(lowetrue)..psfenergies(highetrue)] best fit the interval [E_low..E_high]. It is assumned that the psf energy vector from PSF file is exponential against its index, or however that E is closer to psfenergies(i) than to psfenergies(i+1) if

$$\frac{E}{psfenergies(i)} < \frac{psfenergies(i+1)}{E}$$

4.3. Evaluation of specwt array as the weight of the energy of each interval based on the spectral index. This array has as many values as psfenergies has, and each of them is evaluated as

$$psfenergies(i)^{1-index} - psfenergies(i+1)^{1-index}$$

The value of $psfenergies(i+1)^{1-index}$ is assumed to be zero for the last available value of i, so the last value of specwt is set to $psfenergies(i)^{1-index}$
This is the implementation of point 2.1.

4.4. Evaluation of edparr array, energy dispersion for each energy channel, based on the EDP input file. Each value of edparr is the sum of EDP(energy channel, observed energy, theta, phi=0) for all the observed energies in the interval we are considering.
This is the implementation of point 2.2.

4.5. Evaluation of psfArr (to be normalized in the next step). For each rho, the corresponding value of psfArr is evaluated as the sum on all the values of the Energy of the product of the following values:
- PSF(rho, psi(=0), theta, phi(=0), E)
- AEFF(E, theta, phi(=0))
- edparr(E) as from step 4
- specwt(E) as from step 3
This is the implementation of point 2.3.

4.6. Normalization of psfArr. If $area(\varrho) = 2\pi\Delta\varrho \cdot sin(\varrho)$ is the area of the annulus defined in section 1, the array psfArr is scaled so that $\sum_{\rho=0}^{\rho_{max}} psfArr(\varrho) \cdot area(\varrho) = 1$.
Actually, each value of psfArr is multiplied only by $sin(\varrho)$ and the sum of all this, multiplied by $2\pi\Delta\varrho$ is the normalization value, where $\Delta\varrho = rhoArr(1) - rhoArr(0)$.

4.7. The array rhos from PSF file used so far is assigned to the corresponding outpur function parameter, and psfArr is returned.

5. Changes since the last version

Two bugs were fixed:

5.1. Not all the energy intervals were considered while evaluating edparr, and however they were assigned to the wrong array bin. The new code:

```
for (int etrue = 0; etrue < numpsfenergies; etrue++)
   for (int eobs = lowetrue;  eobs <= highetrue; eobs++)
      edparr(etrue) += AlikeEdpGridClass::Val(psfenergies(etrue), psfenergies(eobs), theta, 0.0);
```

replaced the previous:

```
for (etrue = lowetrue; etrue <= highetrue ; etrue++)
   for (eobs = lowetrue;  eobs <= numpsfenergies-1; eobs++)
      edparr(eobs) += AlikeEdpGridClass::Val(psfenergies(etrue), psfenergies(eobs), theta, 0.0);
```

5.2. The last value of specwt and defined in 1.1, and of PSF did not include the last energy channel.

6. AlikePsfMap::CalcWeighted source code

```
/// Find the closest index for a logarithmic scale
static int FindBestIndex(const TVectorD& energyArr, long energyCount, Float_t E)
{
int best = energyCount-1;
if (energyArr(energyCount-1) >= E) {
   for (best=0; energyArr(best)<E; best++);
   if (best>0 && E / energyArr(best-1) < energyArr(best) / E)
      best--;
   }
return best;
}

TVectorD AlikePsfArray::CalcWeighted(
     Float_t   index,   // IN: Energy spectral index
     Float_t   E\_low,  // IN: Lowest considered energy (application option)
     Float_t   E\_high, // IN: Highest considered energy (application option)
     Float_t   theta,   // IN: angle from instrument axis
     TVectorD* rho      // OUT: rho values as from PFS file
     )
{
```

```
if (index < 0)
   index = -index;

/// Dati prelevati da file PSF
TVectorD psfenergies = AlikePsfGridClass::energies();
long numpsfenergies = psfenergies.GetNoElements();
TVectorD rhoArr = AlikePsfGridClass::rhos();
long numrhos = rhoArr.GetNoElements();

/// Calcolo degli indici piu' vicini a E_low e E_high in una scala logaritmica
int lowetrue = FindBestIndex(psfenergies, numpsfenergies, E_low);
int highetrue = FindBestIndex(psfenergies, numpsfenergies, E_high);

/// Calcolo del peso di ogni energia in base all'indice spettrale
TVectorD specwt(numpsfenergies);
for (int i=0; i < numpsfenergies-1; i++)
   specwt(i) = pow(psfenergies(i) , (1.0 -index)) - pow(psfenergies(i+1), (1.0 -index)) ;
specwt(numpsfenergies-1) = pow(psfenergies(numpsfenergies-1) , (1.0 -index));

/// Calcolo della dispersione energetica totale per ogni canale di energia
TVectorD edparr(numpsfenergies);
for (int etrue = 0; etrue < numpsfenergies; etrue++)
   for (int eobs = lowetrue;  eobs <= highetrue; eobs++)
      edparr(etrue) = AlikeEdpGridClass::Val(psfenergies(etrue), psfenergies(eobs), theta, 0.0);

/// Calcolo della psf da normalizzare
TVectorD psfArr(numrhos);
for (int etrue = 0; etrue<numpsfenergies; etrue++) {
   double energyDep =
      AlikeAeffGridClass::Val(psfenergies(etrue),theta,0.0) * edparr(etrue) * pecwt(etrue);
   for (int i = 0; i < numrhos; i++)
      psfArr(i) += energyDep * likePsfGridClass::Val(rhoArr(i),0.0,theta,0.0,psfenergies(etrue));
   }

/// Normalizzazione della psf secondo l'area di ogni corona circolare
double dsum = 0.0;
for (int i=0; i<numrhos; i++)
   dsum += psfArr(i) * sind(rhoArr(i));
double deltaRho = rhoArr(1)-rhoArr(0);
psfArr *= 1.0 / (dsum * 2 * PI * deltaRho);

/// Restituzione del vettore dei rho effettivamente utilizzato nel calcolo appena terminato
rho->ResizeTo(rhoArr);
*rho = rhoArr;
return psfArr;
}
```