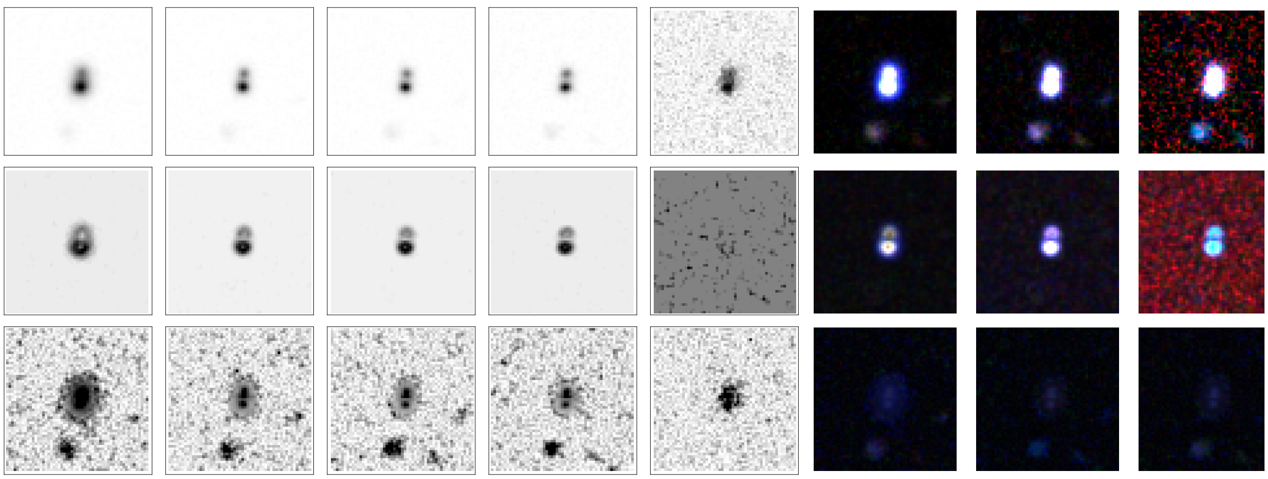
```
(*
A bunch of functions, you will NOT need the Sersic ones,
i.e. Sersic or flase. The others, in particular Gint and flatrot,
are the core of the module. Gint is a Gaussian, with widths dx and dy in pixels,
centered on (0,0). flatrot takes two position coordinates (x,y),
plus an andle "pa", and a flattening "flat", and computes the rotated elliptical
 radius. You'll need them when placing Gaussian blobs on top of images.
*)
bs[n_] = 2. * n - 1 / 3.;
Sersic[R_, n_] = ℯ^(-bs[n] R^(1/n));
flase[x_, y_, flat_, pa_, n_] =
  Sersic[(flat * ( x * Cos[pa] + y * Sin[pa] )^2 + ( -Sin[pa] * x + Cos[pa] * y )^2 / flat )^0.5, n];
(*this is a cylindrical Gaussian in 2D radius x, note the denominator!*)
G[x_, dx_] = ℯ^(-0.5 x^2/dx^2) / ((2 π) dx^2);
(*this is a 2D Gaussian averaged over small pixels*)
Gint[x_, y_, dx_, dy_] = (9. / 16) G[x, dx] G[y, dy] + (3. / 32)
     (G[x + 1, dx] G[y, dy] + G[x - 1, dx] G[y, dy] + G[x, dx] G[y + 1, dy] + G[x, dx] G[y - 1, dy]) +
    (1 / 64.) (G[x - 1, dx] G[y - 1, dy] + G[x - 1, dx] G[y + 1, dy] +
       G[x + 1, dx] G[y - 1, dy] + G[x + 1, dx] G[y + 1, dy]);
(*this computes the elliptical radius*)
flatrot = Compile[{{x, _Real}, {y, _Real}, {flat, _Real}, {pa, _Real}},
    (flat * ( x * Cos[pa] + y * Sin[pa] )^2 + ( -Sin[pa] * x + Cos[pa] * y )^2 / flat )^0.5];
(*e.g. a Gaussian component will be G[flatrot[#1-xcenter,#2-ycenter,flat,pa],sigma],
where #1 and #2 are x and y indices*)

getshapes[Ixx_, Iyy_, Ixy_] := ((*translate Iij second moments into shape parameters*)
   Δ = ((Ixx - Iyy)^2 + 4 Ixy^2)^(1/2);
   ftmp = ((Ixx + Iyy - Δ) / (Ixx + Iyy + Δ))^(1/2);
   phitmp = ArcSin[2 Ixy / Δ] / 2;
   phitmp = Boole[Ixx ≥ Iyy] * phitmp + Boole[Ixx < Iyy] * (Sign[Ixy] π / 2 - phitmp);
   phitmp = Mod[phitmp, 3.14159];
   Rtmp = (Ixx + Iyy)^0.5;
   Return[{Rtmp, phitmp, ftmp}];
  );
paintpsf := ((*partry<--->Join[centers,secmoms,strehl];*)
    (*build nof "model" PSFs in each band, without flux normalisation, return it*)
   (*e.g. a Gaussian component will be G[flatrot[#1-xcenter,#2-ycenter,flat,pa],sigma],
   where #1 and #2 are x and y indices*)
   cores = Array[G[flatrot[#2 - xcenter[[#1]], #3 - ycenter[[#1]], innflas[[#1]], innpas[[#1]]],
       innsigmas[[#1]]] &, {nof, imsize, imsize}];
   wings = Array[G[flatrot[#2 - xcenter[[#1]], #3 - ycenter[[#1]], outflas[[#1]], outpas[[#1]]],
       outsigmas[[#1]]] &, {nof, imsize, imsize}];
   tottot = Array[Total[Total[cores[[#]]]] / Total[Total[wings[[#]]]] &, nof];
   fluxratio = strehl^-1 - 1;
   fluxratio = fluxratio / tottot;
   modelpsf = Array[cores[[#]] + fluxratio[[#]] * wings[[#]] &, nof];
   modelpsf = Array[modelpsf[[#]] / Total[Total[modelpsf[[#]]]] &, nof];
   (*this assigns "model" a core-wing psf, with total flux equalling ONE,
   and returns it so we can use it to paint as many PSFs as needed*)
   Return[modelpsf];
  );
```

```
(*The following lines are used to load the cutouts of a target with
 name 3038925090_?.fits, you'll have to change this in your python code*)
idd = 3 038 925 090;
nameg = ToString[StringForm["``_g.fits", idd]];
namer = ToString[StringForm["``_r.fits", idd]];
namei = ToString[StringForm["``_i.fits", idd]];
namez = ToString[StringForm["``_z.fits", idd]];
nameY = ToString[StringForm["``_Y.fits", idd]];
imgg = First@Import[nameg, "RawData"];
imgr = First@Import[namer, "RawData"];
imgi = First@Import[namei, "RawData"];
imgz = First@Import[namez, "RawData"];
imgY = First@Import[nameY, "RawData"];
(*this is the length of each sied of each DES cutout*)
imsize = Length[imgg];
red = 1.25 imgi / 3.;
green = 1.25 imgr / 2.5;
blue = 1.25 imgg;
rgb =
  Array[{red[[1 + imsize - #1, #2]], green[[1 + imsize - #1, #2]], blue[[1 + imsize - #1, #2]]} &,
   {imsize, imsize}];
(*this is needed in case you want to save the cutouts in a pdf file*)
outname = ToString[StringForm["DES``.pdf", idd]];

(*
this packs the grizY cutouts in a "file" array, of length "nof".
  In python, you'd need to initialize a nof-size-size array,
where "size" is the length of each side of a cutout
*)
file = {imgg, imgr, imgi, imgz, imgY};
nof = Length[file];(*For example, here "nof" is 5*)
imsizes = Array[Length[file[[#]]] &, nof];
(*this cuts the central part of the grizY cutouts.
  To be on the safe side, I'd recommend replacing "cuts" with 1 and "cutw" wth imsizes[[1]]
*)
cuts = Floor[imsizes[[1]] / 4];
cutw = Floor[imsizes[[1]] / 2];
file = Array[file[[#1]][[cuts + #2 - 1, cuts + #3 - 1]] &, {nof, cutw, cutw}];
imsizes = Array[Length[file[[#]]] &, nof];
pixsizes = {0.263, 0.263, 0.263, 0.263, 0.263};
(*
the following is a QnD way of estimating sky brightness and noise in each of the
 grizY cutouts, you should actually use a median or a trimmed mean (google them!).
  You'll need the backg and bcknoise in the Em routine to place point sources,
excluding regions that have low S/N. Here skypix,
backg, bcknoise are arrays of length "nof".
*)
fovs = imsizes / 4;
skypix = Array[
   Sum[UnitStep[(ki - Floor[imsizes[[#]] / 2])^2 + (ki - Floor[imsizes[[#]] / 2])^2 - fovs[[#]]^2],
     {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] &, nof];
backg = Array[Sum[file[[#, ki, kj]] UnitStep[(ki - Floor[imsizes[[#]] / 2])^2 +
          (ki - Floor[imsizes[[#]] / 2])^2 - fovs[[#]]^2],
       {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] / skypix[[#]] &, nof];
bcknoise = Array[(Sum[(file[[#, ki, kj]] - backg[[#]])^2
          UnitStep[(ki - Floor[imsizes[[#]] / 2])^2 + (ki - Floor[imsizes[[#]] / 2])^2 - fovs[[#]]^2],
         {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] / skypix[[#]])^0.5 &, nof];
```

```
(*
from the raw data cutouts above,
build the "smoothed" data and an estimate of the noise maps. To do so,
you'll use a convolution with a 3-by-3 smoothing kernel "smint". The
  coefficients of "smint" have a reason, but I'll tell you separately.
*)
smint = {{1 / 64., 3 / 32., 1 / 64.}, {3 / 32., 9 / 16., 3 / 32.}, {1 / 64., 3 / 32., 1 / 64.}};
delta = {{0., 0., 0.}, {0., 1., 0.}, {0., 0., 0.}};
data = Array[ListConvolve[smint, file[[#]] - backg[[#]]] &, nof];
imsizes = Array[Length[data[[#]]] &, nof];
(*rawdata=Array[ListConvolve[delta,file[[#]]-backg[[#]]]&,nof];*)
derr = Array[ListConvolve[smint, (file[[#]] - backg[[#]])^2] &, nof];
derr = Array[derr[[#]] - data[[#]]^2 &, nof];
derr = Array[derr[[#]]^0.5 &, nof];
imsize = imsizes[[1]];
rgb = Array[{(1.25 / 3.) data[[3]][[1 + imsizes[[3]] - #1, #2]],
      (1.25 / 2.5) data[[2]][[1 + imsizes[[2]] - #1, #2]],
      1.25 data[[1]][[1 + imsizes[[1]] - #1, #2]]} &, {imsizes[[1]], imsizes[[1]]}];
rgb2 = Array[{(1.25 / 3.) data[[4]][[1 + imsizes[[4]] - #1, #2]],
      (1.25 / 2.5) data[[3]][[1 + imsizes[[3]] - #1, #2]],
      1.25 data[[2]][[1 + imsizes[[2]] - #1, #2]]} &, {imsizes[[1]], imsizes[[1]]}];
rgb3 = Array[{(1.25 / 3.) data[[5]][[1 + imsizes[[5]] - #1, #2]],
      (1.25 / 2.5) data[[4]][[1 + imsizes[[4]] - #1, #2]],
      1.25 data[[3]][[1 + imsizes[[3]] - #1, #2]]} &, {imsizes[[1]], imsizes[[1]]}];
rgbres1 = Array[{(1.25 / 3.) derr[[3]][[1 + imsizes[[3]] - #1, #2]],
      (1.25 / 2.5) derr[[2]][[1 + imsizes[[2]] - #1, #2]],
      1.25 derr[[1]][[1 + imsizes[[1]] - #1, #2]]} &, {imsizes[[1]], imsizes[[1]]}];
rgbres2 = Array[{(1.25 / 3.) derr[[4]][[1 + imsizes[[4]] - #1, #2]],
      (1.25 / 2.5) derr[[3]][[1 + imsizes[[3]] - #1, #2]],
      1.25 derr[[2]][[1 + imsizes[[2]] - #1, #2]]} &, {imsizes[[1]], imsizes[[1]]}];
rgbres3 = Array[{(1.25 / 3.) derr[[5]][[1 + imsizes[[5]] - #1, #2]],
      (1.25 / 2.5) derr[[4]][[1 + imsizes[[4]] - #1, #2]],
      1.25 derr[[3]][[1 + imsizes[[3]] - #1, #2]]} &, {imsizes[[1]], imsizes[[1]]}];
signoise = Array[Abs[trunc[data[[#1, #2, #3]] / derr[[#1, #2, #3]]]] &, {nof, imsize, imsize}];
rgbsn1 = Array[
    {(1.25 / 3) signoise[[3, 1 + imsize - #1, #2]], (1.25 / 2.5) signoise[[2, 1 + imsize - #1, #2]],
      1.25 signoise[[1, 1 + imsize - #1, #2]]}  &, {imsize, imsize}];
rgbsn2 = Array[{(1.25 / 3) signoise[[5, 1 + imsize - #1, #2]],
      (1.25 / 2.5) signoise[[4, 1 + imsize - #1, #2]],
      1.25 signoise[[3, 1 + imsize - #1, #2]]}  &, {imsize, imsize}];
rgbsn3 = Array[{(1.25 / 3) signoise[[4, 1 + imsize - #1, #2]],
      (1.25 / 2.5) signoise[[3, 1 + imsize - #1, #2]],
      1.25 signoise[[2, 1 + imsize - #1, #2]]}  &, {imsize, imsize}];
(*put a lower level to the estimated noise, when it's too low*)
ifi = 1;
While[ifi ≤ nof, {
    derr[[ifi]] = Array[
      Max[{derr[[ifi]][[#1, #2]], bcknoise[[ifi]]}] &, {imsizes[[ifi]], imsizes[[ifi]]}];
    ifi +=
     1}];
```

```
(*the first plot that you should build,
to make sure that everything works fine: the first row has the data in grizY bands,
plus (g,r,i), (r,i,z), (i,z,Y) colour-composites. The second row is the same
  but fr the estimated noise. The third one is for the signal-to-noise ratio.
   I've used coefficients that let you see something in the colour-composites,
feelfree to choose your own.*)
GraphicsGrid[{

  Join[Table[ArrayPlot[data[[kb]]], {kb, 1, nof}],
   {Graphics[Raster[rgb / 50]], Graphics[Raster[rgb2 / 50]], Graphics[Raster[rgb3 / 50]]}]
  ,
  Join[Table[ArrayPlot[20 derr[[kb]]], {kb, 1, nof}], {Graphics[Raster[rgbres1 / 50]],
    Graphics[Raster[rgbres2 / 50]], Graphics[Raster[rgbres3 / 50]]}]
  ,
  Join[Table[ArrayPlot[signoise[[kb]]], {kb, 1, nof}], {Graphics[Raster[rgbsn1 / 20]],
    Graphics[Raster[rgbsn2 / 20]], Graphics[Raster[rgbsn3 / 20]]}]

 }]
```



```
(*some quick flux calibrations: in DES, the magnitudes are as simple as in "bandmags",
which again is an array of length "nof"*)
bandmags = Array[-2.5 (Log[Total[Total[data[[#]]]]] / Log[10.] - 9) &, nof];
minmags = bandmags - 0.5;
maxmags = bandmags + 2.5 * 1;
minflux = 10^(9.-0.4 maxmags);
maxflux = 10^(9.-0.4 minmags);
minflux = 3 bcknoise;

(*now do the 1PSF fit*)

(*one Gaussian fit*)

centroids = Array[{imsizes[[#]] / 2., imsizes[[#]] / 2.} &, nof];
centroids = Array[{imsizes[[#]] / 2., imsizes[[#]] / 2.} &, nof];
Iner = ConstantArray[{{3.^2, 0.}, {0., 3.^2}}, nof];
irec = 1;
Nrecpsf = 2;

Miner = Array[PseudoInverse[Iner[[#]]] &, nof];
(*if you want to avoid summing over all pixels, put some convenient limits*)
kilo = Array[Floor[centroids[[#]][[1]] - 10.] &, nof];
kihi = Array[Floor[centroids[[#]][[1]] + 10.] &, nof];
kjlo = Array[Floor[centroids[[#]][[2]] - 10.] &, nof];
kjhi = Array[Floor[centroids[[#]][[2]] + 10.] &, nof];
(*adaptive moments below need the prefctor 2 because of ws in the sum*)
```

```
(*Gaussian windowing function*)
ws = Array[Table[ⅇ^(-0.5 Miner[[#]][[1, 1]] (ki - centroids[[#]][[1]])^2 -
        0.5 Miner[[#]][[2, 2]] (kj - centroids[[#]][[2]])^2 -
        Miner[[#]][[1, 2]] (ki - centroids[[#]][[1]]) (kj - centroids[[#]][[2]])),
     {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] &, nof];
psfws = Array[Table[data[[#, ki, kj]] ws[[#, ki, kj]] UnitStep[data[[#, ki, kj]]],
     {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] &, nof];
nws = Array[Total[Total[psfws[[#]]]] &, nof];
centroids = Array[N[Sum[{ki, kj} psfws[[#]][[ki, kj]],
      {ki, kilo[[#]], kihi[[#]]}, {kj, kjlo[[#]], kjhi[[#]]}] / nws[[#]]] &, nof];
Iner = Array[2. * Sum[{{(ki - centroids[[#]][[1]])^2, (ki - centroids[[#]][[1]]) *
          (kj - centroids[[#]][[2]])}, {(ki - centroids[[#]][[1]]) *
          (kj - centroids[[#]][[2]]), (kj - centroids[[#]][[2]])^2}} psfws[[#]][[ki, kj]],
      {ki, kilo[[#]], kihi[[#]]}, {kj, kjlo[[#]], kjhi[[#]]}] / nws[[#]] &, nof];

(*print the resulting centroids, to make sure that everything is going smoothly*)
centroids
```

```
{{28.027, 28.679}, {28.0928, 28.7582},
 {28.1246, 28.8485}, {28.0607, 28.8862}, {27.8013, 28.5149}}
```

```
(*now adjust the centroids and "Iner" matrices recursively*)
While[irec < Nrecpsf + 1, {
   Miner = Array[PseudoInverse[Iner[[#]]] &, nof];
   kilo = Array[Floor[centroids[[#]][[1]] - 10.] &, nof];
   kihi = Array[Floor[centroids[[#]][[1]] + 10.] &, nof];
   kjlo = Array[Floor[centroids[[#]][[2]] - 10.] &, nof];
   kjhi = Array[Floor[centroids[[#]][[2]] + 10.] &, nof];
   ws =
    Array[Table[ⅇ^(-0.5 Miner[[#]][[1, 1]] (ki - centroids[[#]][[1]])^2 - 0.5 Miner[[#]][[2, 2]]
          (kj - centroids[[#]][[2]])^2 - Miner[[#]][[1, 2]] (ki - centroids[[#]][[1]])
          (kj - centroids[[#]][[2]])), {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] &, nof];
   psfws = Array[Table[data[[#, ki, kj]] ws[[#, ki, kj]] UnitStep[data[[#, ki, kj]]],
      {ki, 1, imsizes[[#]]}, {kj, 1, imsizes[[#]]}] &, nof];
   nws = Array[Total[Total[psfws[[#]]]] &, nof];
   centroids = Array[N[Sum[{ki, kj} psfws[[#]][[ki, kj]],
        {ki, kilo[[#]], kihi[[#]]}, {kj, kjlo[[#]], kjhi[[#]]}] / nws[[#]]] &, nof];
   Iner = Array[2. * Sum[{{(ki - centroids[[#]][[1]])^2, (ki - centroids[[#]][[1]]) *
            (kj - centroids[[#]][[2]])}, {(ki - centroids[[#]][[1]]) *
            (kj - centroids[[#]][[2]]), (kj - centroids[[#]][[2]])^2}} psfws[[#]][[ki, kj]],
        {ki, kilo[[#]], kihi[[#]]}, {kj, kjlo[[#]], kjhi[[#]]}] / nws[[#]] &, nof];
   (*adaptive moments need the prefctor 2 because
    of
    ws
    in
    the
    sum*)

   irec++}];

centroids
```

```
{{28.4521, 29.3044}, {28.5142, 29.1505},
 {28.5758, 29.2332}, {28.4488, 29.2805}, {27.9634, 29.1468}}
```

```
Iner

{{{13.3053, -0.820087}, {-0.820087, 6.48511}},
 {{11.03, -0.644235}, {-0.644235, 3.66658}}, {{11.3137, -0.724066}, {-0.724066, 3.36157}},
 {{11.6871, -0.84113}, {-0.84113, 3.40496}}, {{14.8819, -1.11849}, {-1.11849, 7.61607}}}

(*these will be used to initialize the 2PSF
 fit: they give the width of the blob in the grizY bands. Again,
"psfsigmas" has length "nof"*)
psfsigmas = Array[(Iner[[#]][[1, 1]] + Iner[[#]][[2, 2]])^0.5 &, nof] * 1.4142 / 2

{3.14563, 2.71074, 2.70878, 2.74697, 3.35392}
```

```
(*these give you a width (twice as that),
a p.a. and a b/a estimate for the blob in each band*)
pars1psf = Array[getshapes[Iner[[#]][[1, 1]], Iner[[#]][[2, 2]], Iner[[#]][[1, 2]]] &, nof]
```

```
{{4.44864, 3.02359, 0.690376}, {3.83361, 3.05498, 0.570698},
 {3.83083, 3.05152, 0.53821}, {3.88485, 3.04139, 0.531101}, {4.7432, 2.99226, 0.703465}}
```

```
(*build "nof" fake PSF's, which will be the blobs we'll use to find the best chi^2 with
 one extended source. Originally I was fitting for core, wings and strehl; here,
we'll use basically the same parameters for core and wings, to do it quickly*)
Rcorep = Array[0.99 * pars1psf[[#]][[1]] * 0.71 &, nof];
(*the `R' from getshapes is overestimated by sqrt(2)*)
Rwingp = Array[1.01 * pars1psf[[#]][[1]] * 0.71 &, nof];
pacorep = Array[pars1psf[[#]][[2]] &, nof];
pawingp = Array[pars1psf[[#]][[2]] &, nof];
bacorep = Array[pars1psf[[#]][[3]] &, nof];
bawingp = Array[pars1psf[[#]][[3]] &, nof];
strehlp = ConstantArray[0.501, nof];
psfcore = Array[ConstantArray[0., {imsizes[[#]], imsizes[[#]]}] &, nof];
psfwing = psfcore;
magicwin = 15.;(*this regulates how many pixels you actually want to paint,
using the "kilo"..."kjhi" limits below for each cutout*)
kilo = Array[Floor[centroids[[#]][[1]] - magicwin] &, nof];
kihi = Array[Ceiling[centroids[[#]][[1]] + magicwin] &, nof];
kjlo = Array[Floor[centroids[[#]][[2]] - magicwin] &, nof];
kjhi = Array[Ceiling[centroids[[#]][[2]] + magicwin] &, nof];
(*loop over the "nof" bands and the pixels in each band to paint the blob*)
ifi = 1;
While[ifi ≤ nof, {
   ki = kilo[[ifi]];
   While[ki ≤ kihi[[ifi]], {
     kj = kjlo[[ifi]];
     While[kj ≤ kjhi[[ifi]], {

       psfcore[[ifi]][[ki, kj]] =
        G[flatrot[(ki - centroids[[ifi, 1]]) / bacorep[[ifi]], (kj - centroids[[ifi, 2]]) /
           bacorep[[ifi]], bacorep[[ifi]], pacorep[[ifi]]], Rcorep[[ifi]]];
       psfwing[[ifi]][[ki, kj]] = G[flatrot[(ki - centroids[[ifi, 1]]) / bawingp[[ifi]],
          (kj - centroids[[ifi, 2]]) / bawingp[[ifi]],
          bawingp[[ifi]], pawingp[[ifi]]], Rwingp[[ifi]]];

       kj ++}];
     ki ++}];
   ifi ++}];
(*normalize the blob, as if it were a PSF*)
totcore = Array[Total[Total[psfcore[[#]]]] &, nof];
totwing = Array[Total[Total[psfwing[[#]]]] &, nof];
psftry = Array[
   psfcore[[#]] + (strehlp[[#]]^-1 - 1) * (totcore[[#]] / totwing[[#]]) * psfwing[[#]] &, nof];
totblob = Array[Total[Total[psfcore[[#]]]] &, nof];
(*Here, compute best-fitting one-
 blob fluxes and magnitudes for the grizY bands ("nof" values), and the weighted chi^2*)
OBnum = Array[ Sum[data[[#]][[ki, kj]] * psftry[[#]][[ki, kj]],
      {ki, kilo[[#]], kihi[[#]]}, {ki, kjlo[[#]], kjhi[[#]]}] &, nof];
OBdet = Array[ Sum[psftry[[#]][[ki, kj]]^2, {ki, kilo[[#]], kihi[[#]]},
      {ki, kjlo[[#]], kjhi[[#]]}] &, nof];
OBfluxes = Array[OBnum[[#]] / OBdet[[#]] &, nof];
OBmags = Array[-2.5 (Log[OBfluxes[[#]]] / Log[10.] - 9.0) &, nof];
(*the prefactor data/(data+bcknoise) is used to restrict
 the chi^2 to those regions where there is actually some signal;
the OBwchi2 is an array of length "nof", as usual*)
OBwchi2 = Array[Sum[((data[[#]][[ki, kj]]) / (data[[#]][[ki, kj]] + bcknoise[[#]])) *
        (data[[#]][[ki, kj]] - OBfluxes[[#]] * psftry[[#]][[ki, kj]])^2 / derr[[#]][[ki, kj]]^2,
      {ki, kilo[[#]], kihi[[#]]}, {ki, kjlo[[#]], kjhi[[#]]}] &, nof];
```

```
(*now, use the 1-blob results to initialize the 2-blob results;
we will compute displcements betwee nthe two blobs for each band,
and then combine them using the S/N of each image. Finally,
we'll determine their shape parameters recursively. In the end,
we record the two-blob best-fitting fluxes, the resulting chi^2,
and output some coloured plot with the blob positions drawn on the thing.
*)
(*save the one-blob moments of inertia computed from above*)
I110 = Array[Iner[[#1]][[1, 1]] &, nof];
I120 = Array[Iner[[#1]][[1, 2]] &, nof];
I220 = Array[Iner[[#1]][[2, 2]] &, nof];
(*use the 1PSF results to initialize the 2PSF fit;
start with a circular PSF with smaller width, and compute its second moments*)
midpsf = 6;
xIQ = yIQ = psfsigmas * 1.4142 / 2;
(*psfwidth reduced by sqrt(2)*)
w00 =
  Array[Table[G[flatrot[(kj - midpsf - 1), (ki - midpsf - 1), 0.99, 0.], psfsigmas[[#]] * 0.71],
     {ki, 1, 2 midpsf + 1}, {kj, 1, 2 midpsf + 1}] &, nof] ;
Iw11 = Array[Sum[(ki - midpsf - 1)^2 w00[[#, ki, kj]], {ki, 1, 2 midpsf + 1}, {kj, 1, 2 midpsf + 1}] /
      Total[Total[w00[[#]]]] &, nof];
Iw22 = Array[Sum[(kj - midpsf - 1)^2 w00[[#, ki, kj]], {ki, 1, 2 midpsf + 1}, {kj, 1, 2 midpsf + 1}] /
      Total[Total[w00[[#]]]] &, nof];
Iw12 = Array[Sum[(kj - midpsf - 1) (ki - midpsf - 1) w00[[#, ki, kj]],
       {ki, 1, 2 midpsf + 1}, {kj, 1, 2 midpsf + 1}] / Total[Total[w00[[#]]]] &, nof];
(*here, "scales" was introduced in case the pixel sizes were different across
 different cutous; it's not the case for DES, so we'l just set them to 1.*)
scales = {1., 1., 1., 1., 1.};(*scales[[1]]=1 always*)
(*here we start: compute displacements and widths analytically,
supposing that the big blob is replaced by two circular blobs*)
xsep = Array[(Abs[I110[[#]] - Iw11[[#]]])^0.5 &, nof];
ysep = Array[(Abs[I220[[#]] - Iw22[[#]]])^0.5 * Sign[I120[[#]]] &, nof];
centr1 = Array[{centroids[[#]][[1]] + xsep[[#]], centroids[[#]][[2]] + ysep[[#]]} &, nof];
centr2 = Array[{centroids[[#]][[1]] - xsep[[#]], centroids[[#]][[2]] - ysep[[#]]} &, nof];
(*S/N in each cutout*)
cardsn =
  Array[Sum[data[[#, ki, kj]], {ki, kilo[[#]], kihi[[#]]}, {kj, kjlo[[#]], kjhi[[#]]}] &, nof];
(*center of the first "small" blob*)
meanc1 = Sum[centr1[[kb]] * scales[[kb]] * cardsn[[kb]], {kb, 1, nof - 1}] /
    Sum[cardsn[[kb]], {kb, 1, nof - 1}];(*in pixels, in DES cards*)
(*center of the second "small" blob*)
meanc2 = Sum[centr2[[kb]] * scales[[kb]] * cardsn[[kb]], {kb, 1, nof - 1}] /
    Sum[cardsn[[kb]], {kb, 1, nof - 1}];(*in pixels, in DES cards*)
(*in each card's pixelvalues*)
(*print a few things to make sure that the stuff is working*)
displ = meanc2 - meanc1
```

```
{-1.99251, 2.00462}
```

```
meanc1
```

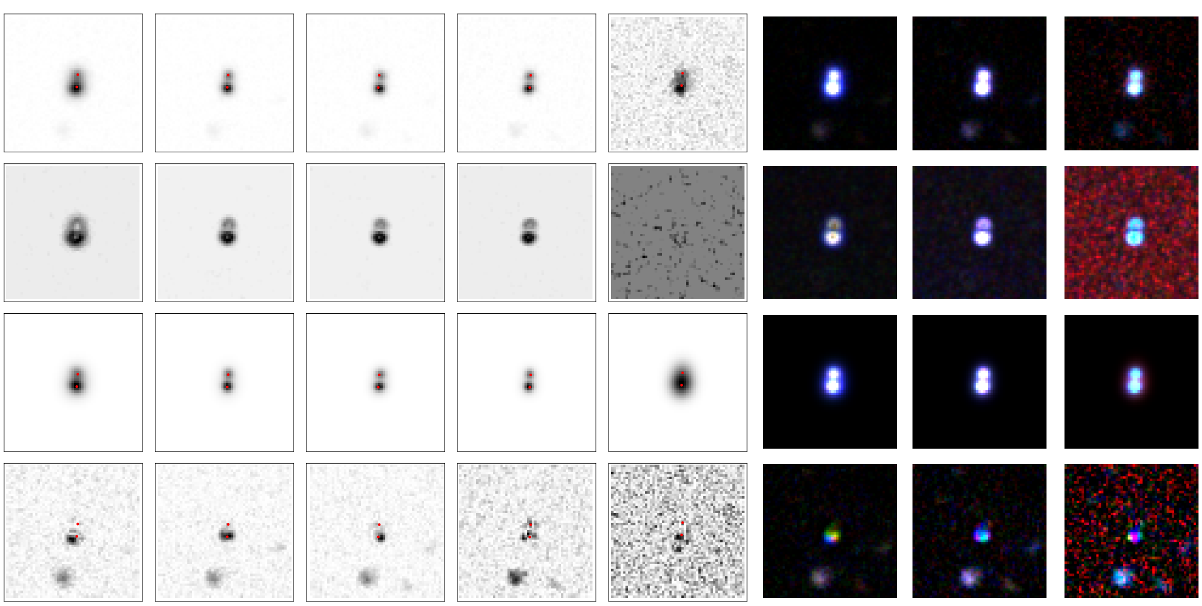
```
{29.7804, 28.2731}
```

```
meanc2

{27.7879, 30.2778}

Mw = Array[PseudoInverse[{{Iw11[[#]], Iw12[[#]]}, {Iw12[[#]], Iw22[[#]]}}] &, nof];
(*new w's*)
w11 = Array[e^(-0.5 (#2 - centr1[[#1]][[1]]) Mw[[#1]][[1, 1]] (#2 - centr1[[#1]][[1]]) -
        0.5 (#3 - centr1[[#1]][[2]]) Mw[[#1]][[2, 2]] (#3 - centr1[[#1]][[2]]) -
        (#2 - centr1[[#1]][[1]]) Mw[[#1]][[1, 2]]
         (#3 - centr1[[#1]][[2]])) &, {nof, imsize, imsize}];
w12 = Array[e^(-0.5 (#2 - centr2[[#1]][[1]]) Mw[[#1]][[1, 1]] (#2 - centr2[[#1]][[1]]) -
        0.5 (#3 - centr2[[#1]][[2]]) Mw[[#1]][[2, 2]] (#3 - centr2[[#1]][[2]]) -
        (#2 - centr2[[#1]][[1]]) Mw[[#1]][[1, 2]]
         (#3 - centr2[[#1]][[2]])) &, {nof, imsize, imsize}];
(*compute tot.fluxes w/TotalTotal*)
totf = Array[{Total[Total[w11[[#]]]], Total[Total[w12[[#]]]]} &, nof];
a11 = a12 = Array[Total[Total[data[[#]]]] / 2 &, nof];

irec = 1;

(*the following is an Expectation-Maximization algorithm that
  adjusts the centroids and shape parameters of the two little blobs;
at the end of each iteration, the shape parameters fo the two blobs are combined
 to ensure that both blobs in the next iteration share the same shape parameters*)

Nrec = 20;
twocenrec = Nrec;
cont = True;
While[irec ≤ Nrec && cont, {
   kilo = Array[Floor[centroids[[#]][[1]] - 10.] &, nof];
   kihi = Array[Floor[centroids[[#]][[1]] + 10.] &, nof];
   kjlo = Array[Floor[centroids[[#]][[2]] - 10.] &, nof];
   kjhi = Array[Floor[centroids[[#]][[2]] + 10.] &, nof];
   (*windowing functions*)
   T1 = Array[
     UnitStep[(#2 - kilo[[#1]]) (kihi[[#1]] - #2)] UnitStep[(#3 - kjlo[[#1]]) (kjhi[[#1]] - #3)] *
       a11[[#1]] * w11[[#1]][[#2, #3]] / (a11[[#1]] * w11[[#1]][[#2, #3]] +
          a12[[#1]] * w12[[#1]][[#2, #3]] + bcknoise[[#1]]) &, {nof, imsize, imsize}];
   T2 = Array[UnitStep[(#2 - kilo[[#1]]) (kihi[[#1]] - #2)] UnitStep[(#3 - kjlo[[#1]])
         (kjhi[[#1]] - #3)] * a12[[#1]] * w12[[#1]][[#2, #3]] / (a11[[#1]] * w11[[#1]][[#2, #3]] +
          a12[[#1]] * w12[[#1]][[#2, #3]] + bcknoise[[#1]]) &, {nof, imsize, imsize}];
   (*new centroids*)
   normw1 = Array[Sum[UnitStep[(ki - kilo[[#1]]) (kihi[[#1]] - ki)]
         UnitStep[(kj - kjlo[[#1]]) (kjhi[[#1]] - kj)] * T1[[#]][[ki, kj]]
         data[[#]][[ki, kj]], {ki, 1, imsize}, {kj, 1, imsize}] &, nof];
   centr1 = Array[Sum[{ki, kj} UnitStep[(ki - kilo[[#1]]) (kihi[[#1]] - ki)]
          UnitStep[(kj - kjlo[[#1]]) (kjhi[[#1]] - kj)] * T1[[#]][[ki, kj]] data[[#]][[ki, kj]],
         {ki, 1, imsize}, {kj, 1, imsize}] / normw1[[#]] &, nof];
   normw2 = Array[Sum[UnitStep[(ki - kilo[[#1]]) (kihi[[#1]] - ki)]
         UnitStep[(kj - kjlo[[#1]]) (kjhi[[#1]] - kj)] * T2[[#]][[ki, kj]]
         data[[#]][[ki, kj]], {ki, 1, imsize}, {kj, 1, imsize}] &, nof];
```

```
centr2 = Array[Sum[{ki, kj} UnitStep[(ki - kilo[[#1]]) (kihi[[#1]] - ki)]
        UnitStep[(kj - kjlo[[#1]]) (kjhi[[#1]] - kj)] * T2[[#]][[ki, kj]] data[[#]][[ki, kj]],
       {ki, 1, imsize}, {kj, 1, imsize}] / normw2[[#]] &, nof];

(*griz flux-weight the centroids!*)
tota1 = Sum[a11[[kb]], {kb, 1, nof}];
tota2 = Sum[a12[[kb]], {kb, 1, nof}];
meanc1 = Sum[centr1[[kb]] a11[[kb]], {kb, 1, nof}] / tota1;
meanc2 = Sum[centr2[[kb]] a12[[kb]], {kb, 1, nof}] / tota2;
centr1 = ConstantArray[meanc1, nof];
centr2 = ConstantArray[meanc2, nof];
(*same covariance matrix for both point-sources*)
Mw = Array[PseudoInverse[{{Iw11[[#]], Iw12[[#]]}, {Iw12[[#]], Iw22[[#]]}}] &, nof];
(*new w's*)
w11 = Array[e^(-0.5 (#2 - centr1[[#1]][[1]]) Mw[[#1]][[1, 1]] (#2 - centr1[[#1]][[1]]) -
        0.5 (#3 - centr1[[#1]][[2]]) Mw[[#1]][[2, 2]] (#3 - centr1[[#1]][[2]]) -
        (#2 - centr1[[#1]][[1]]) Mw[[#1]][[1, 2]]
         (#3 - centr1[[#1]][[2]])) &, {nof, imsize, imsize}];
w12 = Array[e^(-0.5 (#2 - centr2[[#1]][[1]]) Mw[[#1]][[1, 1]] (#2 - centr2[[#1]][[1]]) -
        0.5 (#3 - centr2[[#1]][[2]]) Mw[[#1]][[2, 2]] (#3 - centr2[[#1]][[2]]) -
        (#2 - centr2[[#1]][[1]]) Mw[[#1]][[1, 2]]
         (#3 - centr2[[#1]][[2]])) &, {nof, imsize, imsize}];
(*compute tot.fluxes w/TotalTotal*)
totf = Array[{Total[Total[w11[[#]]]], Total[Total[w12[[#]]]]} &, nof];

(*new amplitudes*)
If[irec > twocenrec, {
  (*this stuff here computes the best-fitting fluxes of two blobs,
  which can be done by solving a linear equation*)
  A12 = Array[Sum[w11[[#]][[ki, kj]] w12[[#]][[ki, kj]] / derr[[#1, ki, kj]]^2,
       {ki, 1, imsize}, {kj, 1, imsize}] &, nof];
  Akl = Array[{{Sum[w11[[#]][[ki, kj]]^2 / derr[[#1, ki, kj]]^2,
          {ki, 1, imsize}, {kj, 1, imsize}], A12[[#]]},
       {A12[[#]], Sum[(*UnitStep[clip-Abs[tabdev[[#]][[ki,kj]]]]*)w12[[#]][[ki, kj]]^2 /
           derr[[#1, ki, kj]]^2, {ki, 1, imsize}, {kj, 1, imsize}]}} &, nof];
  Bk = Array[{Sum[w11[[#]][[ki, kj]] data[[#]][[ki, kj]] / derr[[#1, ki, kj]]^2, {ki,
          1, imsize}, {kj, 1, imsize}], Sum[(*UnitStep[clip-Abs[tabdev[[#]][[ki,kj]]]]*)
        w12[[#]][[ki, kj]] data[[#]][[ki, kj]] / derr[[#1, ki, kj]]^2,
        {ki, 1, imsize}, {kj, 1, imsize}]} &, nof];
  ampl = Array[PseudoInverse[Akl[[#]]].Bk[[#]] &, nof];

  (*adjust minimum fluxes*)
  a11 = Array[Max[{ampl[[#]][[1]], minflux[[#]] / totf[[#]][[1]]}] &, nof];
  a12 = Array[Max[{ampl[[#]][[2]], minflux[[#]] / totf[[#]][[2]]}] &, nof];
  Print["a11=", a11, "    , a12=", a12];
  Nefftwo = Array[Total[Total[T1[[#]] + T2[[#]]]] &, nof];
  tabdev = Array[(a11[[#1]] * w11[[#1]][[#2, #3]] + a12[[#1]] * w12[[#1]][[#2, #3]] -
```

```
            data[[#1]][[#2, #3]])^2 / derr[[#1, #2, #3]]^2 &, {nof, imsize, imsize}];
    (*you'll need to change this a bit: the chi^s in each band must
       be weighted by data/(data+bcknoise)*)
    chi21 = Array[Sum[tabdev[[#]][[ki, kj]], {ki, 1, imsize}, {kj, 1, imsize}] &, nof];

    Print["chi21=", chi21];
   }];

  irec++;
 }];
xseptmp = meanc2[[1]] - meanc1[[1]];
yseptmp = meanc2[[2]] - meanc1[[2]];
displ = meanc2 - meanc1;
```

```
(*in the end, you should be able to print something like the following... here
 "modelpic" (not defined above) is an array of shape [nof,imsizes[[1]],imsizes[[1]]],
i.e. "nof" model cutotus that are the two blobs with their best-
 fit fluxes. The final plot is: data (with blob positions overlaid), noise, model,
residuals. In each line, the cutouts are as above: g,r,i,z,Y, and RGB colour-composites.*)
overplot = Table[
   Show[ArrayPlot[data[[kb]]],
    ListPlot[{{centr1p[[kb]][[2]] - 0.5, 0.5 + imsizes[[kb]] - centr1p[[kb]][[1]]},
      {centr1p[[kb]][[2]] + displp[[2]] - 0.5, 0.5 + imsizes[[kb]] - centr1p[[kb]][[1]] -
        displp[[1]]}}, PlotStyle → Directive[Red, PointSize[0.01]]]
   ],
   {kb, 1, nof}];
overmod = Table[
   Show[ArrayPlot[modelpic[[kb]]],
    ListPlot[{{centr1p[[kb]][[2]] - 0.5, 0.5 + imsizes[[kb]] - centr1p[[kb]][[1]]},
      {centr1p[[kb]][[2]] + displp[[2]] - 0.5, 0.5 + imsizes[[kb]] - centr1p[[kb]][[1]] -
        displp[[1]]}}, PlotStyle → Directive[Red, PointSize[0.01]]]
   ],
   {kb, 1, nof}];
overres = Table[
   Show[ArrayPlot[N[data[[kb]] - modelpic[[kb]]]],
    ListPlot[{{centr1p[[kb]][[2]] - 0.5, 0.5 + imsizes[[kb]] - centr1p[[kb]][[1]]},
      {centr1p[[kb]][[2]] + displp[[2]] - 0.5, 0.5 + imsizes[[kb]] - centr1p[[kb]][[1]] -
        displp[[1]]}}, PlotStyle → Directive[Red, PointSize[0.01]]]
   ],
   {kb, 1, nof}];
synopsis = GraphicsGrid[{

   Join[overplot,
    {Graphics[Raster[rgb / 100]], Graphics[Raster[rgb2 / 100]], Graphics[Raster[rgb3 / 150]]}]
   ,
   Join[Table[ArrayPlot[derr[[kb]]], {kb, 1, nof}], {Graphics[Raster[rgbres1 / 50]],
     Graphics[Raster[rgbres2 / 50]], Graphics[Raster[rgbres3 / 50]]}]
   ,
   Join[overmod, {Graphics[Raster[rgbmod1 / 100]],
     Graphics[Raster[rgbmod3 / 100]], Graphics[Raster[rgbmod2 / 150]]}]
   ,
   Join[overres, {Graphics[Raster[-N[rgbmmd1] / 40]],
     Graphics[Raster[-N[rgbmmd3] / 40]], Graphics[Raster[-N[rgbmmd2] / 40]]}]
  }]
```



```
(*In the end, the following should be saved as lines in a
 log file: the one-blob magnitudes, the one-blob weighted chi^2's,
the two-blob magnitudes, the two-blob weighted chi^2's. Once this is done,
we'll use those outputs to retain just those
 systems whose magnitudes and chi^2's satisfy some properties*)
```