Measuring Detected Source Extent Using Mexican-Hat Optimization

Version 0.7

September 27, 2007

John C. Houck

Kavli Institute for Astrophysics and Space Science,
Massachusetts Institute of Technology, Cambridge MA 02139

1. Motivation

In Level 3 pipeline processing of the *Chandra* archive, source detection is currently performed by *wavdetect* (Freeman et al. 2002). The primary output product of *wavdetect* is a list of elliptical regions indicating the position and extent of each detected source. Unfortunately, the *wavdetect* source positions often are not precisely centered on the corresponding count distribution and the point source region sizes do not smoothly track the *PSF*. The purpose of this memo is to describe a more reliable algorithm for defining elliptical source regions for automatically detected sources. The algorithm described in §2 was derived in part from the techniques described by Damiani et al. (1997).

2. Algorithm

The 2-D correlation integral of a function $w(x, y; \boldsymbol{\alpha})$ with another function $f(x, y)$ may be written in the form

$$W(x, y; \boldsymbol{\alpha}) = \int_{-X}^{X} \int_{-Y}^{Y} \mathrm{d}x' \mathrm{d}y' \; w(x' - x, y' - y; \boldsymbol{\alpha}) \; f(x', y'), \tag{1}$$

where the region of interest is $|x| \leq X$ and $|y| \leq Y$ and where $\boldsymbol{\alpha}$ represents a vector of parameters.

To detect sources in an image, it is of interest to consider the correlation integral of the image with an elliptical "Mexican-Hat" function which may be written in the dimensionless form

$$w(x, y; \boldsymbol{\alpha}) \equiv \left(2 - \rho^2\right) \exp\left(-\rho^2/2\right), \tag{2}$$

where

$$\rho^2 \equiv \frac{1}{a_1^2}(x \cos\phi + y \sin\phi)^2 + \frac{1}{a_2^2}(-x \sin\phi + y \cos\phi)^2 \tag{3}$$

and $\boldsymbol{\alpha} = (a_1, a_2, \phi)$.

In this context, the Mexican-Hat function is often called a *wavelet* and the associated correlation integral is often called a *wavelet transform*. A similar transform is an important component of *wavdetect*.

When $f(x,y)$ contains a single elliptical Gaussian peak with position, size and orientation specified by the parameters $(x_0, y_0, \sigma_1, \sigma_2, \phi_0)$, the quantity

$$\psi(x,y;\boldsymbol{\alpha}) \equiv W(x,y;\boldsymbol{\alpha})/(a_1 a_2)^{1/2} \tag{4}$$

has a maximum at (x_0, y_0) for $a_i = \sigma_i\sqrt{3}$ and $\phi = \phi_0$ (Damiani et al. 1997)[1]. It follows that one can determine the position, size and orientation of the elliptical Gaussian peak in $f(x,y)$ by determining the five parameters that maximize ψ. For simplicity, I will refer to this technique for determining source region parameters as *Mexican-Hat Optimization* (*MHO*).

To see the effect of a linear background, consider the value of $\psi_0(\boldsymbol{\alpha}) \equiv \psi(x_0, y_0; \boldsymbol{\alpha})$ when $f(x,y)$ has the form

$$f(x,y) = c_0 + c_1 x + c_2 y + c_3 xy + S(x - x_0, y - y_0), \tag{5}$$

where the c_k are constants and where $S(x,y)$ is a peaked function representing a source. Without loss of generality, one can choose coordinates in which the peak is centered at the origin and aligned with the coordinate axes so that $\phi = 0$. In this coordinate system,

$$\begin{aligned}\psi_0(\boldsymbol{\alpha}) &= W(0,0;\boldsymbol{\alpha})/(a_1 a_2)^{1/2} \\ &= \frac{1}{(a_1 a_2)^{1/2}} \int_{-X}^{X} \mathrm{d}x \int_{-Y}^{Y} \mathrm{d}y \, \left\{ \left[2 - x^2/a_1^2 - y^2/a_2^2\right] \exp\left(-\frac{x^2}{2a_1^2}\right) \exp\left(-\frac{y^2}{2a_2^2}\right) \right\} \\ &\qquad \times \left[c_0 + c_1 x + c_2 y + c_3 xy + S(x,y)\right], \end{aligned} \tag{6}$$

where the limits of integration have been chosen to be symmetric about the origin. Because the quantity in braces is an even function of x and y, terms linear in x and y vanish. The factor multiplying c_0 can be expressed analytically as a sum of terms involving error functions and terms $\propto \exp(-X^2/2a_1^2)$ and $\propto \exp(-Y^2/2a_2^2)$ (see Freeman et al. 2002). The error function terms cancel and, in the limit that $X \gg a_1$ and $Y \gg a_2$, the exponential terms rapidly approach zero. It follows that $\psi_0(\boldsymbol{\alpha})$ is dominated by the peaked function, $S(x,y)$, and is insensitive to the presence of a linear background.

3. Implementation

In practical applications, $f(x,y)$ is usually an image of a source and has discrete pixel values f_{mn} at each sky position (x_m, y_n). Equation (1) should then be expressed as a discrete sum of the form

$$W(x_i, y_j; \boldsymbol{\alpha}) = \sum_{m,n} w_{mn}(x_i, y_j; \boldsymbol{\alpha})\, f_{mn}, \tag{7}$$

where the $w_{mn}(\boldsymbol{\alpha})$ are determined by integrating $w(x,y,\boldsymbol{\alpha})$ over the area of each pixel and where the sum extends over all pixels of the image. Integration of $w(x,y,\boldsymbol{\alpha})$ from equation (2) does not yield

[1]When $f(x,y)$ contains a circular disk of radius R with constant surface brightness, ψ has a maximum at (x_0, y_0) for $a_1 = a_2 = R/\sqrt{2}$.

a simple closed-form solution and computational expense makes numerical integration relatively cumbersome. In practice, for the purpose of optimizing $\psi_0(\boldsymbol{\alpha})$, a rectangular approximation for the integral over each pixel seems sufficient. In this approximation,

$$w_{mn}(x_i, y_j; \boldsymbol{\alpha}) \approx w(x_m - x_i, y_n - y_j; \boldsymbol{\alpha}) \Delta x \Delta y, \tag{8}$$

where $w(x, y; \boldsymbol{\alpha})$ is evaluated at the center of each pixel and where the pixel area is $\Delta x \Delta y$.

When sources are closely spaced, accurate estimates of the source size and position are especially important. In the context of the Level 3 detect pipeline, I assume that an approximate source position has been determined by *wavdetect*. A small sub-image of the source is extracted using these coordinates. The accuracy of this source position is refined by searching the center of the sub-image for the coordinates (x_0, y_0) that maximize $\psi_0(a, a, 0)$. Once the source center coordinates (x_0, y_0) have been established, a new sub-image is extracted using the improved source position.

A refined estimate of the source size is obtained by computing $\psi_0(a, a, 0)$ on a grid of a values spanning the half-width of the source image f_{mn}. When a single source is present, the location of a local maximum, where $\partial_a \psi_0 = 0$ and $\partial_a^2 \psi_0 \leq 0$, provides a good estimate of the source size. When the source of interest is blended with other nearby sources, an inflection point, where $\partial_a^2 \psi_0 = 0$, often occurs near the "edge" of the central source. The smallest a that corresponds to either a local maximum or an inflection point provides a good initial guess for the source size. When the first occurrence of $\partial_a \psi_0 = 0$ occurs at a local minimum, where $\partial_a^2 \psi_0 > 0$, the smallest value of a on the pixel grid is used as the initial source size.

Using these refined estimates of the source position and size, the size and orientation of an elliptical Gaussian source region are derived by maximizing $\psi_0(\boldsymbol{\alpha}) \equiv \psi(x_0, y_0; \boldsymbol{\alpha})$. Because optimization algorithms are usually designed to search for minima, $\psi_0(\boldsymbol{\alpha})$ can be maximized by minimizing $-\psi_0(\boldsymbol{\alpha})$. The minimization can be performed by any generic minimization algorithm; a prototype implementation of this algorithm used the *subplex* minimization algorithm (Rowan 1990).

4. Tests

For the *MHO* algorithm (§2), the primary sources of error are Poisson noise, pixelization, non-Gaussian structure in the point-spread function (*PSF*), and confusion with nearby point sources (effectively a non-linear background). Monte Carlo simulations using *isis* and *MARX* have been used to examine these issues.

4.1. Ideal *PSF* (no *PSF* blur)

The *MHO* algorithm (§2) was applied to Monte Carlo simulations of elliptical Gaussian sources including Poisson noise, a flat, diffuse background and no *PSF* blur. The simulated sources had semi-major axes $1.125 \leq \sigma_1 \leq 4$ pixels and position angles $0 \leq \phi \leq \pi$ with the semi-minor axis fixed at $\sigma_2 = 1$ pixel. Figure 1 shows the cumulative semi-major axis error distributions for Monte

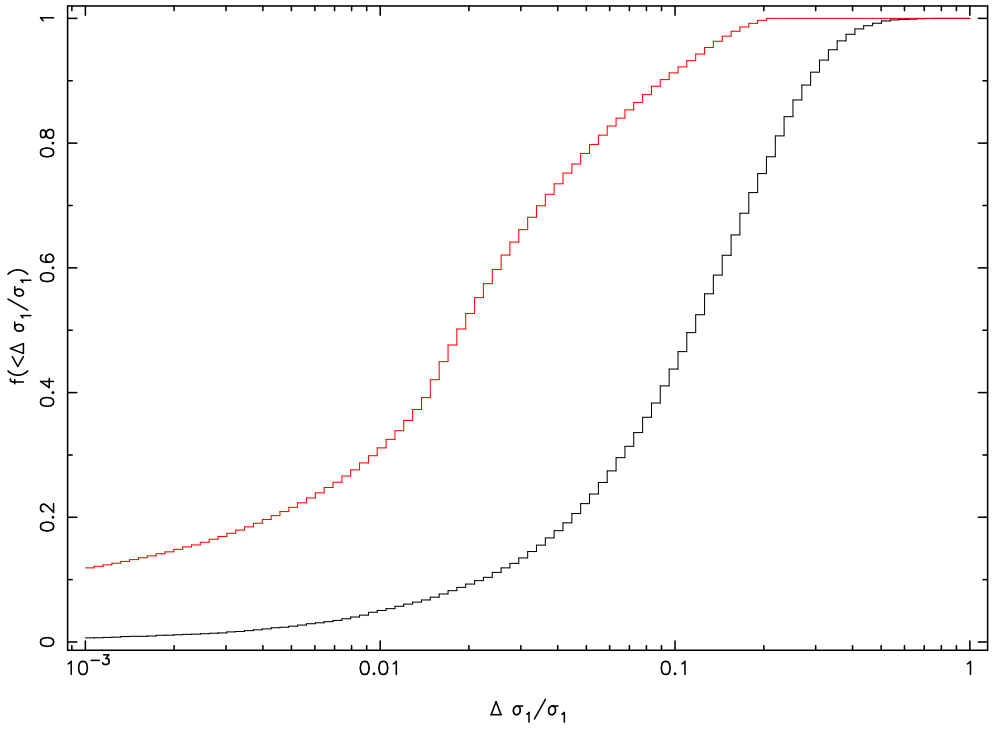


Fig. 1.— Cumulative distribution of semi-major axis fractional errors from Monte Carlo trials with and without Poisson noise. The fake sources had ~ 50 counts each. The red curve represents 5760 trials without Poisson noise, finely sampling the parameter range $1.125 \leq \sigma_1 \leq 4$ pixels and $0 \leq \phi \leq \pi$ with the semi-minor axis fixed at $\sigma_2 = 1$ pixel. The black curve represents 6600 trials including Poisson noise; the parameter ranges were more coarsely sampled and 100 trials were done with each set of parameters.

Carlo trials with ~ 50 count extended sources.

With Poisson noise turned off, 5760 simulations were performed, on a uniform grid of 64 semi-major axis values and 90 position angles. In 68% of the trials, the input semi-major axis σ_1 values were recovered to $\leq$3.2% accuracy, limited primarily by pixelization errors.

With Poisson noise turned on, 100 trials were done with each set of parameter values on a uniform grid of 6 semi-major axis values and 11 position angles. In 68% of the trials, the input σ_1 values were recovered to within $\leq 17\%$ (13%) accuracy for sources with $\sim$50 (100) counts each.

4.2. **MARX** *PSF*

Two *MARX* simulations demonstrate the ability of the *MHO* algorithm (§2) to distinguish between point sources and extended sources when position-dependent, non-Gaussian *PSF* blur is

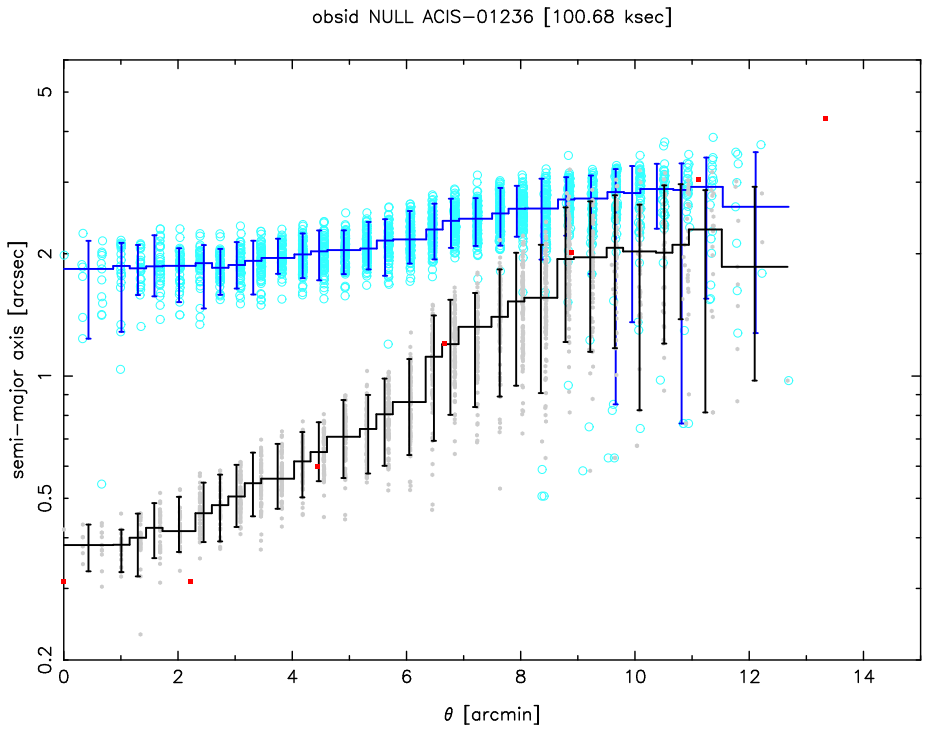


Fig. 2.— Source ellipse semi-major axis, σ, determined by the *MHO* algorithm vs. off-axis angle, θ, for *MARX*-simulated sources (ACIS-I + *Chandra* blank-sky background). The black symbols represent point sources. The blue symbols represent extended sources (Gaussian, $\sigma = 2$ arcsec). The median σ value in each θ bin is plotted with error bars that enclose the central 90% of the σ distribution within each bin. Both source populations have ~ 130 counts per source. For reference, red squares indicate the *ECF*=0.40 radius of the *Chandra PSF* (Allen, Jerius & Gaetz 2004).

present.

To avoid source blends, fake sources were inserted into a source-free uniform background constructed from the ACIS blank-sky background files[2]. Fake sources were evenly spaced in a uniform pattern of concentric rings centered on the optical axis. One simulation contains a pattern of point sources while the other contains a pattern of Gaussian sources having $\sigma = 2$ arcsec. Both source populations have ~ 130 counts per source.

The *MHO* algorithm (§2) was then applied to the list of source positions used in constructing the simulation. The resulting source region size distributions are shown in Figure 2. In both of these tests, the spread in σ values for the extended sources is consistent with the level of scatter seen in Monte Carlo tests without *PSF* blur. This suggests that the primary effect of the *PSF* blur is to introduce a corresponding systematic increase in the σ value obtained by the algorithm. For a Gaussian *PSF* and a Gaussian source, the expected source size is

$$\sigma^2 = \sigma_0^2 + \sigma_{\rm psf}^2, \tag{9}$$

[2]http://cxc.harvard.edu/ciao/threads/acisbackground

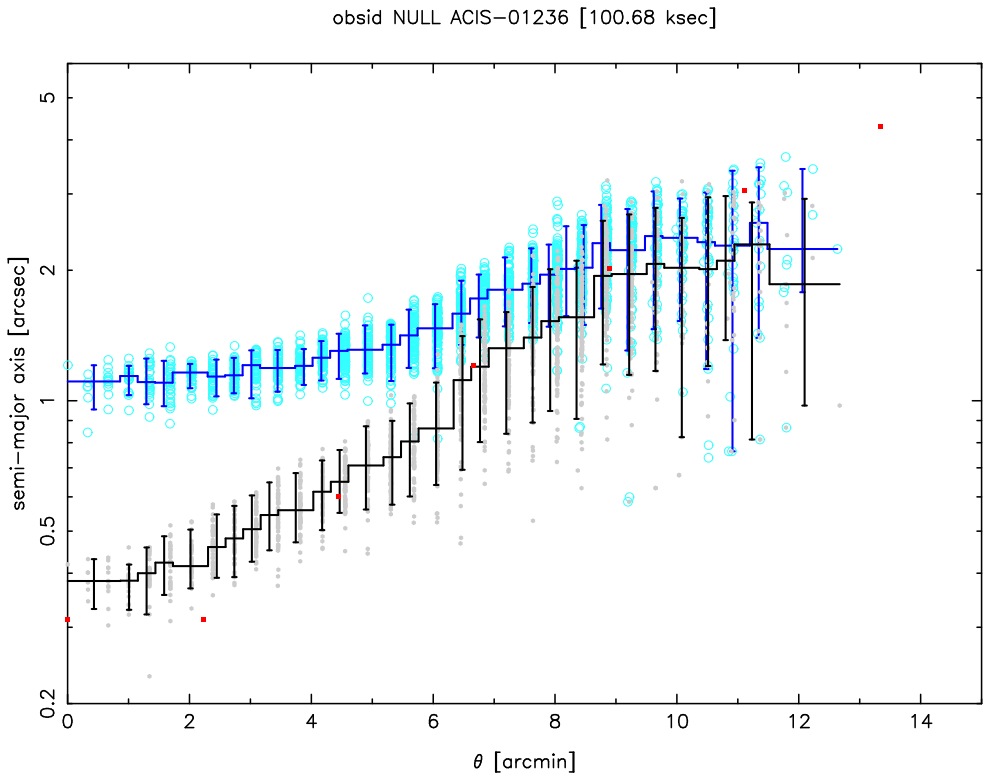


Fig. 3.— Source ellipse semi-major axis, σ, determined by the *MHO* algorithm vs. off-axis angle, θ, for *MARX*-simulated sources (ACIS-I + *Chandra* blank-sky background). The black symbols represent point sources. The blue symbols represent extended sources (uniform surface brightness disk with 2 arcsec radius). The median σ value in each θ bin is plotted with error bars that enclose the central 90% of the σ distribution within each bin. Both source populations have ~ 130 counts per source. For reference, red squares indicate the *ECF*=0.40 radius of the *Chandra PSF* (Allen, Jerius & Gaetz 2004).

where σ_0 is the true source size and $\sigma_{\rm psf}$ is the *PSF* size. Because the *Chandra PSF* is asymmetric and has non-Gaussian components, equation (9) is only approximately correct.

Inspection of Figure 2 shows that the extended sources are unambiguously identified within $\lesssim 8$ arcmin of the optical axis. At larger off-axis angles, the *PSF* size is comparable to the source extent and the two source populations overlap. Most of the outlier points correspond to cases where the simulated source overlaps the edge of the detector.

Near the optical axis, point sources yielded $\sigma \approx 0.4$ arcsec[3] while Gaussian extended sources yielded $\sigma \approx 1.9$ arcsec. For the extended sources, the input source size was $\sigma = 2$ arcsec (4 pixels). Accounting for the ~ 0.4 arcsec blur expected due to the *MARX PSF* and the dither, the expected detector image size is $(2^2 + 0.4^2)^{1/2} = 2.04$ arcsec. The *MHO* algorithm slightly underestimated the Gaussian source size, but the correct value falls within the 1σ error bar.

[3]For a 2-D Gaussian, the fraction, f, of the total flux enclosed within radius r, is $f = 1 - \exp(-r^2/2\sigma^2)$ so that $r = \sigma$ encloses a fraction, $f = 0.393$.

As an additional test, I processed a second test pattern of extended sources, this time using uniform surface brightness disks instead of Gaussians. The radius of each disk source was 2 arcsec, the same as the σ value used for the Gaussian sources. The disk sources had the same flux as the Gaussian sources, so their surface brightness is somewhat higher. Figure 3 compares the distribution of region sizes for the disks with that of the point sources. Near the optical axis, the disk sources yielded semi-major axes of $\sigma \approx 1.1$ arcsec. This apparent underestimate occurs primarily because my implementation of the *MHO* algorithm assumes that the target source is a Gaussian. If the source were known to be a uniform disk, the estimated source size would have been $R = a\sqrt{2} \approx 2.7$ arcsec (the value of a is the same in each case).

4.2.1. Source Region Parameter Uncertainty

To characterize the errors associated with the source region parameters determined by the *MHO* algorithm, I used *MARX* simulations to perform a number of Monte-Carlo trials. Each *MARX* simulation consisted of a test pattern of sources, each source having the same size and approximately the same brightness. I considered point sources with ∼15-300 counts and Gaussian extended sources having $\sigma =$ 1-8 arcsec and ∼15-300 counts within 1σ.

Figure 4 shows the distribution of source region sizes obtained from each of these simulations. In each case, approximately 90% of the source sizes were within $\Delta\sigma$ of the median value where

$$\frac{\Delta\sigma}{\sigma} \equiv f_{\rm mho} \approx 0.3 \left(\frac{15}{N_\sigma}\right)^{1/2}, \tag{10}$$

where N_σ is the number of counts within 1σ.

Figure 5 shows the amplitude of the position error, $r_\varepsilon \equiv (\delta x^2 + \delta y^2)^{1/2}$, as a function of off-axis angle θ. The position errors are distributed such that about 90% fall inside a circle of radius

$$r_\varepsilon \approx 2\bar{\sigma}\, N_\sigma^{-1/2}\, \exp\left(\frac{\theta}{24'}\right) \tag{11}$$

where $\bar{\sigma} \equiv (\sigma_1 + \sigma_2)/2$, N_σ is the number of counts above background within 1σ, and θ is the off-axis angle in arc-minutes. Although equation (11) holds for $N_\sigma \lesssim 300$, it may break down for $N_\sigma \gg 300$. The *MHO* algorithm has not yet been tested extensively in the limit $N_\sigma \gg 300$.

When the source size is comparable to the *PSF* size, the direction of the position error, $\phi_\varepsilon \equiv \tan^{-1}(\delta y/\delta x)$, is correlated with the position angle, ϕ, of the source ellipse major axis (and ϕ is strongly correlated with the position angle of the *PSF* ellipse). The strength of this correlation depends on the number of source counts but is apparent even for relatively faint point sources. For circular extended sources larger than a few arc-seconds in apparent size, the correlation effectively disappears.

For sources comparable in size to the *PSF*, it may be desirable to define a position error ellipse that accounts for the correlation between the position error direction and the position angle of the

source region ellipse. Perhaps the simplest approach is to define a position error ellipse having the same axis ratio and orientation as the source region ellipse, but with semi-axes scaled to match equation (11). The semi-axes of that position error ellipse are

$$\varepsilon_1 = \frac{r_\varepsilon}{(1+\xi^2)^{1/2}}, \qquad \varepsilon_2 = \xi\varepsilon_1, \tag{12}$$

where $r_\varepsilon^2 = \varepsilon_1^2 + \varepsilon_2^2$, and where $\xi \equiv \sigma_2/\sigma_1$, is the axis ratio of the source region ellipse. For sources with $\sigma_i \gg \sigma_{\rm psf}$, the position error direction is not correlated with the orientation of the source region ellipse. In such cases, r_ε properly defines an error circle.

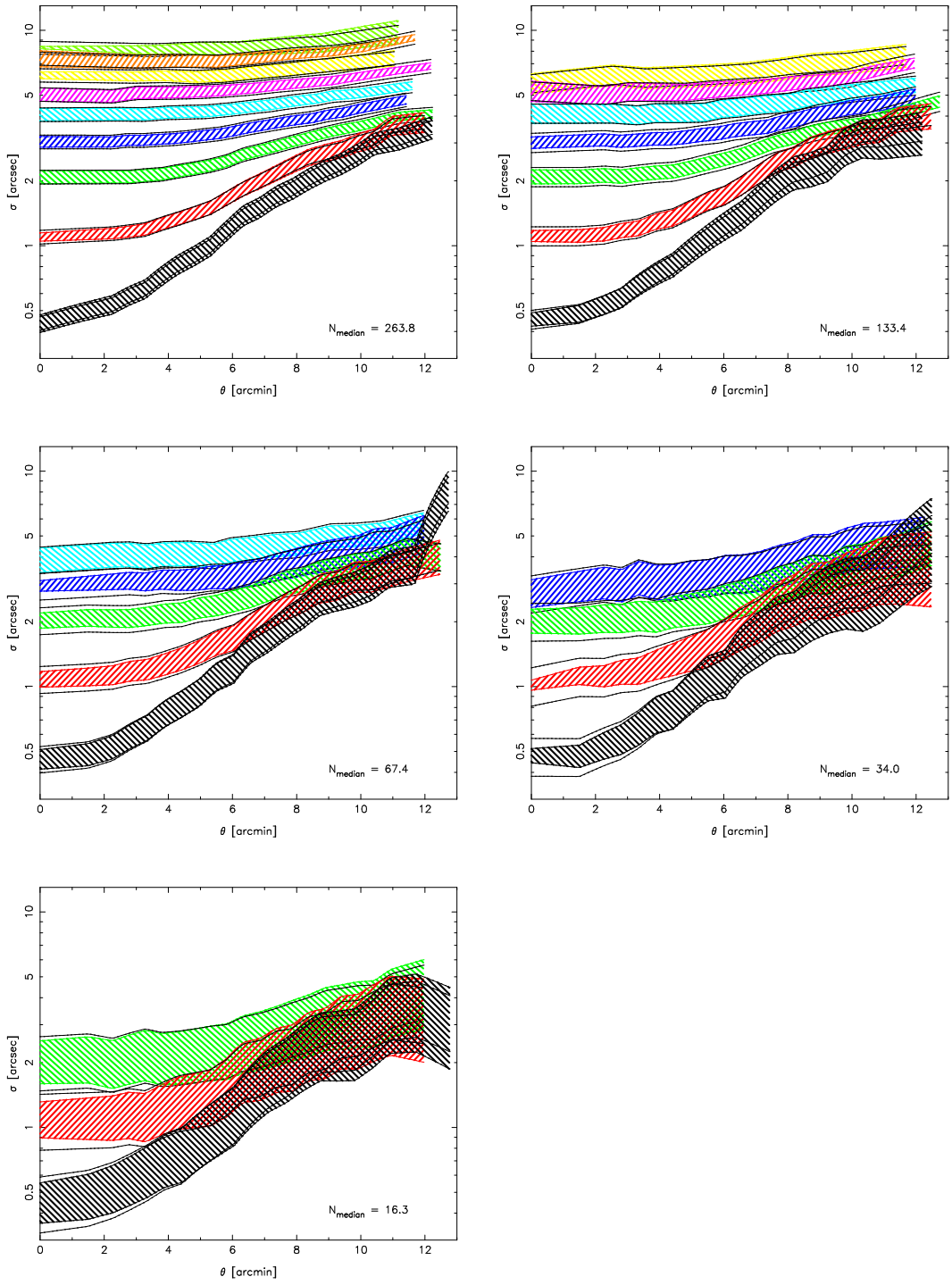


Fig. 4.— Distribution of source region sizes derived from *MARX*-simulated data using the *MHO* algorithm. The simulations included point sources and Gaussian extended sources with σ = 1-8 arcsec. Each panel shows sources having a similar number of counts. The cross-hatched regions approximately represent the central 90% of each source size distribution; point-sources are shown in black. The black lines enclosing each cross-hatched region represent the approximate error bar from equation (10).

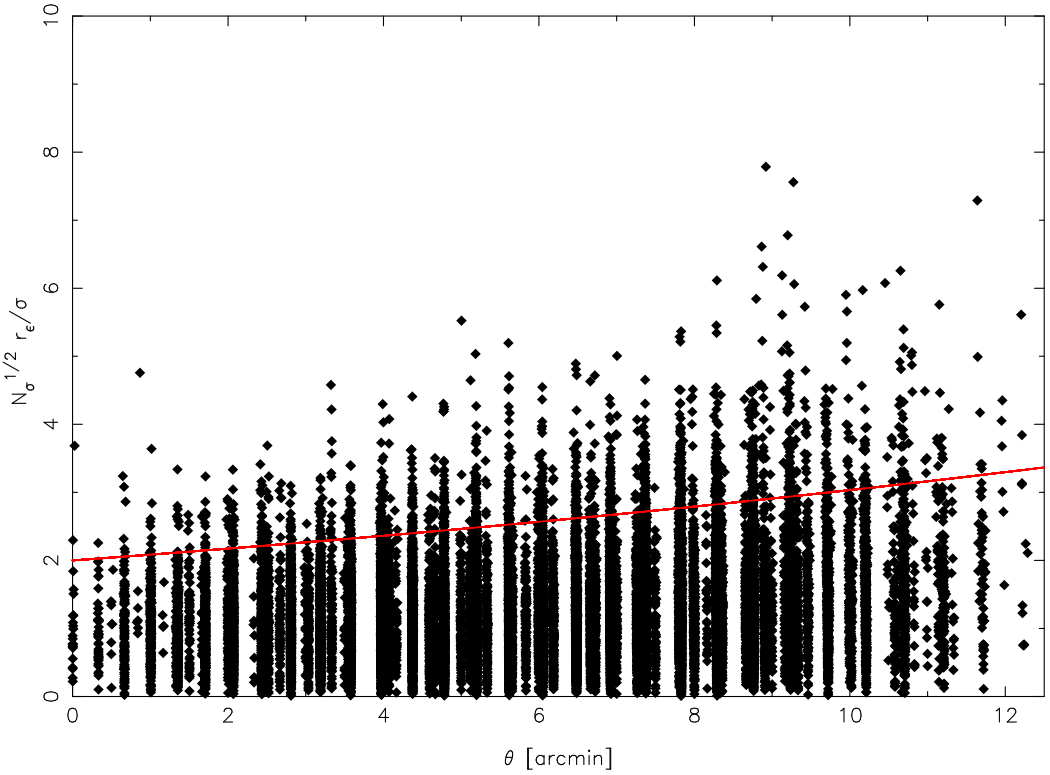


Fig. 5.— Distribution of errors in source position, $N_\sigma^{1/2} r_\varepsilon/\bar{\sigma}$, vs. off-axis angle, θ. Black symbols represent *MARX*-simulated sources spanning a range in brightness and spatial extent. The sample includes both point sources and Gaussian extended sources with $\sigma = 1$–8 arcsec and spans a range in brightness such that the sources have $N_\sigma \approx 15$–300 counts. The red curve corresponds to the expression in equation (11).

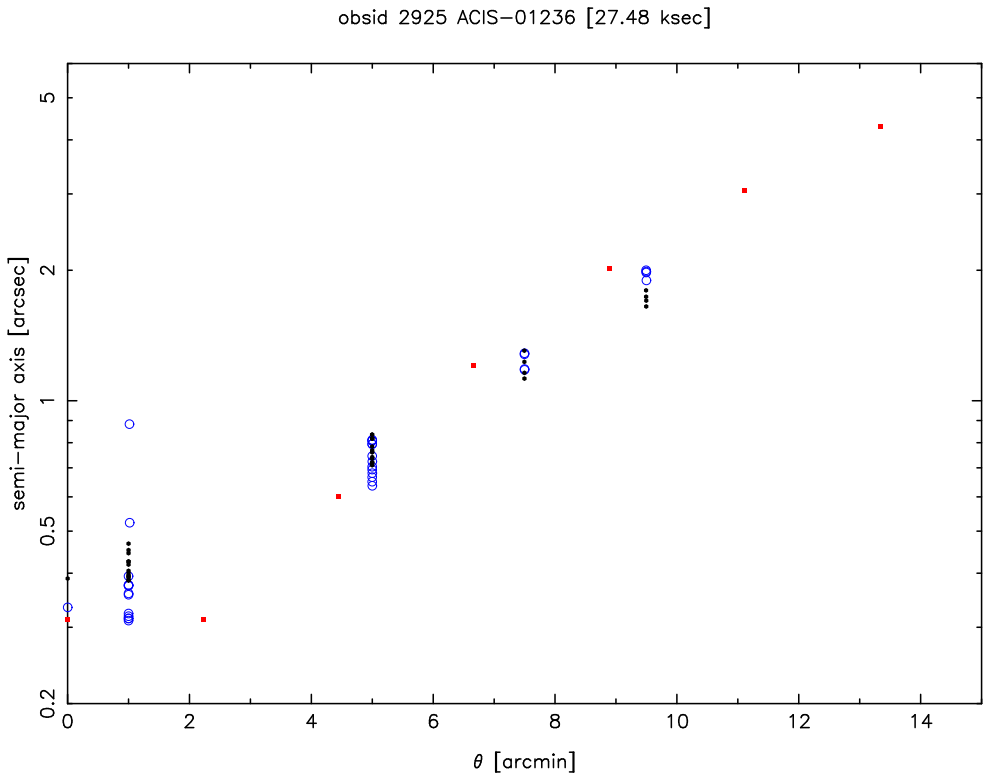
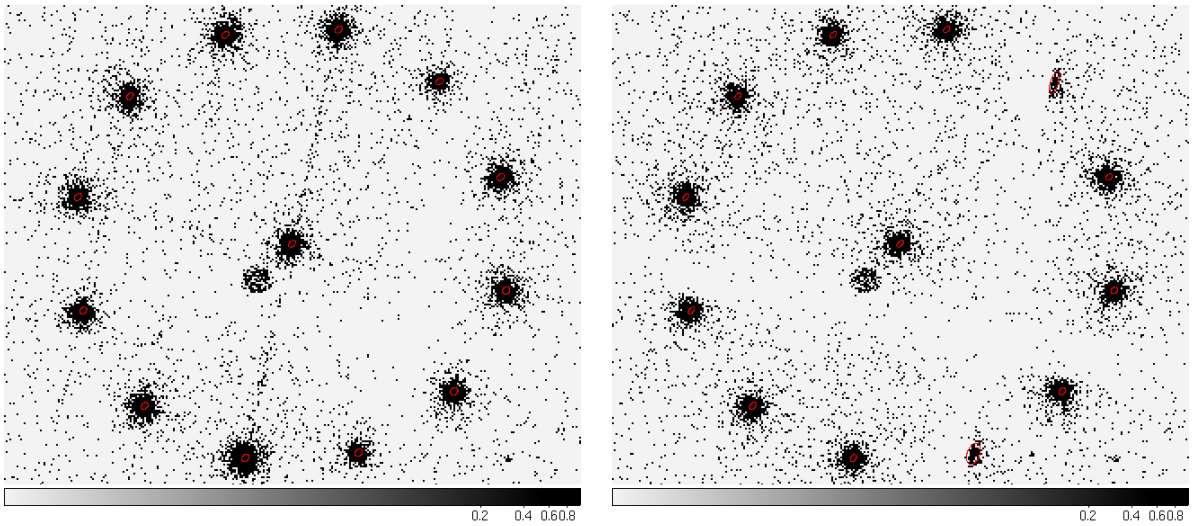


Fig. 6.— Source ellipse semi-major axis, σ, determined by the *MHO* algorithm vs. off-axis angle, θ. Blue symbols represent *SAOSAC*-simulated point sources; black symbols represent *MARX*-simulated point sources. In each case, the simulated point sources were added an event file from *Chandra* obsid 2925. The two outlier points in blue at $\theta = 1$ arcmin correspond to a minor flaw in the *SAOSAC* simulation (see Figure 7). For reference, red squares indicate the *ECF*=0.40 radius of the *Chandra PSF* (Allen, Jerius & Gaetz 2004).

4.3. SAOSAC *PSF*

To compare the performance of the *MHO* algorithm on *SAOSAC*-simulated data with its performance on *MARX*-simulated data, both *MARX* and *SAOSAC* were used to simulate the same test pattern of point sources and the *MHO* algorithm was applied to both data sets (the *SAOSAC* simulation was created by Frank Primini). The resulting distribution of source region semi-major axis values is shown in Figure 6.

Inspection of Figure 6 shows that, aside from two outlier points near $\theta = 1$ arcmin, the *MHO* algorithm generates source regions for both the *MARX* and *SAOSAC* simulations that are almost identical. There does appear to be a slight trend, with *MARX* sources having a systematically flatter size distribution. Nevertheless, these differences are within the error bars expected in most practical applications.



(a) (b)

Fig. 7.— Part of the test pattern of point sources used to compare the performance of the *MHO* algorithm on (a) *MARX*-simulated data and (b) *SAOSAC*-simulated data. In each figure, red ellipses indicate source regions generated by the *MHO* algorithm. Note that these images show all detected events, thus over-emphasizing the *PSF* wings in order to show the fainter chip-gap sources in (b). The 4 arcsec radius extended source near the center of the test pattern is a supernova remnant in NGC 6822 (Tennant 2006).

In Figure 6, the two outlier points near $\theta = 1$ arcmin occur because, after the *SAOSAC* simulation was processed by `psf_project_ray`, the events were filtered to remove events with invalid chip coordinates (<1 or > 1024). Figure 7 shows that this filtering removed a significant fraction of the counts from these two point sources. Because the resulting detector images of these two sources is highly asymmetric, it is not surprising that the *MHO* algorithm generated a highly elliptical source region quite unlike the *Chandra PSF* shape at that location.

Given that the other source regions match within the errors, I conclude that *MARX* simulations are adequate for the purpose of characterizing the performance of the *MHO* algorithm.

5. Caveats

The following caveats apply to the *MHO* algorithm as described in §2:

1. The algorithm requires, as input, a rough estimate of the source size (σ). For best results, this estimate should be within a factor of ∼3 of the actual source σ; an underestimate is better than an overestimate. For point sources and moderately extended sources (*e.g.* $\sigma \lesssim 3$ arcsec), use the approximate *PSF* size.

2. The algorithm tends to focus on any bright, sharply peaked source components and is insensitive to any fainter, more diffuse, extended emission that lies nearby. To detect such extended emission, one might compare the counts in an adjacent elliptical annulus with an estimate of the background level.

3. The algorithm is not designed to separate blended sources and is unlikely to generate optimal source regions in such cases.

4. The algorithm makes no attempt to detect cases in which no significant source is present above background within the ellipse that was initially provided by *wavdetect*. In such cases, subsequent optimization of ψ may yield a meaningless result, such as an ellipse of maximum size or an ellipse of random size centered on a noise peak.

5. The algorithm generates source regions with no internal reference to the *PSF*.

6. The algorithm has not been extensively tested with sources having $N_\sigma \gg 300$ or with sources that are heavily piled-up.

6. Issues

wavdetect often provides inaccurate positions for weak or extended sources and often reports multiple positions for a single extended source. To a certain extent, the *MHO* algorithm can correct the source position, but if that correction fails, the resulting region size is unlikely to be accurate.

One potential alternate method for determining source regions is to fit each source image with one or more 2-D elliptical Gaussians plus a background component. Such fits are likely to involve a relatively large number of free parameters; a linear background component introduces 1-3 free parameters and each additional Gaussian profile introduces 4-6 parameters. For sources with a large number of detected counts, the high signal to noise ratio may enable these fits to converge and even to yield useful confidence intervals on the fit parameters. The extent to which this approach is feasible in the low counts limit is unclear.

7. Recommendations

1. Use *wavdetect* to generate a preliminary list of source regions.

2. Use the *MHO* algorithm to improve the *wavdetect* source regions; a *S-Lang* implementation is provided with this document.

3. An isolated source with a significant number of counts above background is likely to be extended if $\min(\sigma_i) > \sigma_t$, where

$$\sigma_t \equiv \frac{\sigma_{\rm psf}}{1 - \max(f_{\rm psf}, f_{\rm mho})}. \tag{13}$$

   In equation (13), $f_{\rm psf} \equiv \Delta\sigma_{\rm psf}/\sigma_{\rm psf}$ is the fractional uncertainty in the *PSF* size, $f_{\rm mho}$ is the fractional uncertainty in the source region size from equation (10), and N_σ is the number of counts above background inside σ_i. Note that the expression for $f_{\rm mho}$ rapidly becomes meaningless when $N_\sigma \lesssim 15$ counts.

4. A point-like source ($\sigma_i < \sigma_t$) may be considered to be extended if the count-rate in an adjacent elliptical annulus exceeds the expected background count-rate by more than 3σ. However, in crowded fields, this criterion may be misleading.

5. Use equation (9) to estimate the "de-convolved" source size.

6. Use equation (10) to estimate the statistical error in the source region size.

7. Use equation (11) to estimate the statistical error in the relative source position. For sources comparable in size to the *PSF*, a position error ellipse may be defined by scaling the source region semi-axes using equation (12). Note that this is an estimate of the relative position error only; the absolute position error must account for the absolute accuracy of the aspect solution, including the roll angle and the off-axis angle.

REFERENCES

Allen, C., Jerius, D. H., & Gaetz, T. J., 2004, in X-Ray and Gamma-Ray Instrumentation for Astronomy XIII. Edited by Flanagan, Kathryn A.; Siegmund, Oswald H. W. Proceedings of the SPIE, Volume 5165, pp. 423-432 (2004)., ed. K. A. Flanagan, O. H. W. Siegmund, Vol. 5165, 423

Damiani, F., Maggio, A., Micela, G., & Sciortino, S., 1997, ApJ, 483, 350

Freeman, P. E., Kashyap, V., Rosner, R., & Lamb, D. Q., 2002, ApJS, 138, 185

Rowan, T., 1990, *Ph.D. thesis*, University of Texas, Austin

Tennant, A. F., 2006, AJ, 132, 1372

This preprint was prepared with the AAS LaTeX macros v5.2.

```
% -*- mode: SLang; mode: fold -*-

%  Author: John C. Houck <houck@space.mit.edu
% Written: July 2007
%
% This algorithm is intended to generate an elliptical source region
% given an approximate source position (sky X,Y) and a crude estimate
% for the source size, such as may be produced by a source detection
% algorithm.
%
% Interface:
%
%  params[] = mho_find_source_extent (events, x0, y0, crude_src_size,
%                                     img_size, bin_factor);
%    params[] = Source region parameters
%                [x0, y0, sigma_1, sigma_2, phi]
%                (x0, y0) = sky X,Y coordinates of the region center [pixels]
%                (sigma_1, sigma_2) = Gaussian sigma values [pixels]
%                phi = Ellipse position angle [radians]
%                        The +X axis is phi=0.
%                        The +Y axis is phi=PI/2.
%      events = Struct_Type containing sky X, Y coordinates for
%               the detected events.
%      x0, y0 = Approximate sky X, Y coordinates for a detected
%               source.
% crude_src_size = Crude estimate of the source size [pixels].
%               For best results, the estimate should be within a
%               factor ~3 of the actual source size.  For point sources
%               and moderately extended sources, use the approximate
%               PSF sigma at (x0, y0) [e.g. radius enclosing 40% of
%               the flux].
%    img_size = Size [pixels] of the region centered on (x0, y0) to
%               be examined by the algorithm.  If img_size=0, the
%               algorithm will choose a value.
%  bin_factor = Integer factor by which the extracted image
%               should be binned.  If bin_factor=0, the algorithm
%               will choose a value.
%
%  Note:  'pixel' means ACIS pixel = 0.492 arcsec.

% Mho_Max_Size_Factor sets the largest size scale and the maximum axis
% ratio for source regions.  If the value is "too large", things
% will run a lot slower (big images) and confusion problems
```

```
% in crowded fields will probably get worse.
variable Mho_Max_Size_Factor = 3.0;

% isis-specific:  silence warnings that "minimization failed"
public define fit_verbose_warn_hook (s){ }

private variable _e;
try (_e)
{
   require ("maplib");
}
catch AnyError:
{
   %print(_e);
};

#ifnexists maplib_meshgrid
private define maplib_meshgrid (y, x) %{{{
{
   variable
     ny = length(y),
     nx = length(x),
     X = Double_Type[ny, nx],
     Y = Double_Type[ny, nx];

   _for (0, ny-1, 1)
     {
        variable j = ();
        X[j,*] = x[*];
     }
   _for (0, nx-1, 1)
     {
        variable i = ();
        Y[*,i] = y[*];
     }

   return Y, X;
}

%}}}
#endif

private define extract_image (t, x0, y0, xsize, ysize, bin_factor) %{{{
```

```
{
   variable
     hwx = xsize/2,
     hwy = ysize/2;

   variable i = where (abs(t.x - x0) < hwx
                       and abs(t.y - y0) < hwy);

   variable f = struct {x, y, x0, y0, image};

   variable
     nx = nint(xsize/bin_factor),
     ny = nint(ysize/bin_factor);

   % odd dimensions work better, assuming that the
   % image is centered on the source of interest

   if ((nx/2)*2 == nx) nx += 1;
   if ((ny/2)*2 == ny) ny += 1;

   f.x = [x0-hwx:x0+hwx:#nx];
   f.y = [y0-hwy:y0+hwy:#ny];
   f.image = histogram2d (t.y[i], t.x[i], f.y, f.x);
   f.x0 = x0;
   f.y0 = y0;
   f.x -= x0;
   f.y -= y0;

   % grid will give pixel center coordinates
   % except in over-flow bin
   variable
     dx = f.x[1] - f.x[0],
     dy = f.y[1] - f.y[0];
   f.x += 0.5*dx;
   f.y += 0.5*dy;

   return f;
}

%}}}

private define elliptical_mh (x, y, pars) %{{{
{
```

```
   variable phi = pars[2];
   variable r2 = sqr((x*cos(phi) + y*sin(phi))/pars[0])
     + sqr((-x*sin(phi) + y*cos(phi))/pars[1]);
   return (2 - r2) * exp (-r2/2);
}

%}}}

private define correlation_integral (x, y, pars, f) %{{{
{
   variable i = where (f.image != 0);
   if (length(i) == 0)
     return 0.0;

   variable
     dx = f.x[1]-f.x[0],
     dy = f.y[1]-f.y[0];

   variable X, Y, w;
   (Y, X) = maplib_meshgrid (f.y - y, f.x - x);
   w = elliptical_mh (X[i], Y[i], pars);

   return sum (w * f.image[i]) * dx * dy;
}

%}}}

private variable Constraint_Info;

private define axes_constraint (stat, p) %{{{
{
   variable f = Constraint_Info;

   % If the source's physical location is known, the correlation
   % maximum will occur in that pixel, so we don't need to compute
   % the other pixels.
   variable dims = array_shape (f.image);
   variable
     Y = f.y[dims[0]/2],
     X = f.x[dims[1]/2];

   variable W_max;
   W_max = correlation_integral (X, Y, p, f);
```

```
   return -W_max / sqrt(p[0]*p[1]);
}

%}}}

private define randomize_near_value (f, list) %{{{
{
   if (list == NULL)
     list = [1:get_num_pars ()];

   foreach (list)
     {
        variable idx = ();
        variable s = get_par_info(idx);

        if (s.freeze or s.fun != NULL or s.tie != NULL)
          continue;

        variable val;

        do
          {
             if (s.value != 0)
               val = s.value * (1.0 + f * grand());
             else
               val = f * grand();
          }
        while (val < s.min or s.max <= val);

        set_par (idx, val);
     }
}
%}}}

private define compute_stat (a) %{{{
{
   variable info;
   set_par ("constraint", [a, a, 0]);
   () = eval_counts (&info);
   return -info.statistic;
}
```

```
%}}}

define sign_change (y) %{{{
{
   variable sy = sign(y);
   variable i = where (sy != shift(sy,1));
   if (length(i) == 0)
     return i;
   variable n = length(y);
   return i[where(i < n-2)];
}

%}}}

define deriv (y) %{{{
{
   return y - shift(y,1);
}

%}}}

private define correlation_scale (f) %{{{
{
   variable
     as = f.x[where(f.x > 1)],
     w = array_map (Double_Type, &compute_stat, as);

   variable w1, i1;
   w1 = deriv(w);
   i1 = sign_change(w1);

   if (length(i1) == 0)
     {
        return as[0];
     }

   variable w2, i2;
   w2 = deriv(w1);
   i2 = sign_change(w2);

   variable ilist = [i1, i2];
   ilist = ilist[array_sort(ilist)];
   variable i = ilist[0];
```

```
   if (w2[i] > 0)
     {
        if (w[0] > w[i])
          {
             return as[0];
          }
        else i = ilist[1];
     }

   return as[i];
}

%}}}

private define optimizer_init () %{{{
{
   % isis-specific hack:  The fit engine needs a spectrum
   % and a model to fit so provide them -- but in this case,
   % the constraint function is what matters.

   delete_data (all_data);
   Minimum_Stat_Err=1.e-30;
   () = define_counts (0.5, 1.5, 1.0, 1.e-10);
   fit_fun ("bin_width(1)");
}

%}}}

define _mho_optimize_axes (f, a_guess, crude_src_size) %{{{
{
   variable a_max_factor = Mho_Max_Size_Factor;

   ifnot (any(f.image > 0))
     return [-1.0, -1.0, 0.0];

   optimizer_init();

   Constraint_Info = f;
   set_fit_constraint (&axes_constraint, ["a1", "a2", "phi"]);

   % correlation_scale almost always gives the best
   % initial estimate for a.  Is there a situation
```

```
% where the input a_guess is consistently better?

variable a_scale = NULL; %a_guess;
if ((a_scale == NULL) || (a_scale < crude_src_size))
  {
     a_scale = correlation_scale (f);
     if (a_scale < crude_src_size)
       a_scale = crude_src_size;
  }

% Prefer a1 >= a2.
% 0.6 is a common Chandra PSF axis ratio.
%
% a_max is set conservatively low to minimize trouble in
% crowded fields.  For that reason, there's some motivation to
% set a_min generously low -- the PSF size provides a
% natural lower bound to filter on and, if the correlation
% scale is relatively large, the generous lower bound provides
% wiggle room to fit highly elliptical source regions.
variable
  phi_start  = 0.0,
  a_min    = a_scale * 0.2,
  a2_start = a_scale * 0.8,
  a1_start = a_scale * 1.2,
  a_max    = a_scale * a_max_factor;

if (a_max <= a1_start)
  {
     throw ApplicationError, "Mho_Max_Size_Factor=$Mho_Max_Size_Factor is too small"$;
  }

set_par ("constraint",
         [a1_start, a2_start, phi_start],
         [0, 0, 0],
         [a_min, a_min, -1.5*PI],
         [a_max, a_max,  1.5*PI]);

set_fit_method ("subplex;maxnfe=500;tol=1.e-4;scale_factor=0.125");
Fit_Verbose = -1;

variable pars, num_loops = 0;
forever
  {
```

```
        % adjust phi
        freeze(1,2); thaw(3);
        () = fit_counts;

        % adjust all params
        thaw (1,2,3);
        variable status = fit_counts();

        % avoid quitting when axes are at their range limits
        pars = get_par ([1,2,3]);
        variable pegged = (abs(1 - max(pars[[0:1]])/a_max) < 0.01
                           || abs(1 - min(pars[[0:1]])/a_min) < 0.01);
        if (status == 0 && pegged == 0)
          break;

        num_loops++;
        if (num_loops >= 5)
          break;

        % re-set the parameters only if we're going to try
        % another fit
        set_par ("constraint", [a1_start, a2_start, PI*urand()]);
        randomize_near_value (0.05, NULL);
      }

    % Always report a1 >= a2
    pars = get_par([1,2,3]);
    if (pars[0] < pars[1])
      {
        pars = pars[[1,0,2]];
        pars[2] += PI/2;
      }
    % Always report 0 <= phi <= PI
    pars[2] = pars[2] mod PI;
    if (pars[2] < 0) pars[2] += PI;

    set_par ("constraint", pars);

    return pars;
}

%}}}
```

```
private define position_size_constraint (stat, p) %{{{
{
   variable f = Constraint_Info;

   variable
     x = p[0],
     y = p[1],
     a = p[2];

   return -correlation_integral (x, y, [a,a,0], f)/a;
}

%}}}

private define improve_source_position (f, crude_src_size, x0_ref, y0_ref, a_ref) %{{{
{
   ifnot (any(f.image > 0))
     return 0;

   optimizer_init ();

   Constraint_Info = f;
   set_fit_constraint (&position_size_constraint, ["x", "y", "a"]);

   variable
     nx = length(f.x),
     ny = length(f.y),
     ox = nx/4,
     oy = ny/4;

   variable
     a_min = 0.5 * crude_src_size,
     a_max = 0.5 * min([f.x[nx-ox-1], f.y[ny-oy-1]]),
     a = min ([crude_src_size, sqrt(a_min * a_max)]);

   % Look for a better source position
   % near the middle of the sub-image
   set_par ("constraint",
            [    0.0,          0.0,          a],
            [      0,            0,          0],
            [f.x[ox],      f.y[oy],      a_min],
            [f.x[nx-ox-1], f.y[ny-oy-1], a_max]);
```

```
   set_fit_method ("subplex;maxnfe=500;tol=1.0e-4;scale_factor=0.125");
   Fit_Verbose = -1;

   variable initial, final;
   () = eval_counts (&initial);
   () = fit_counts (&final);

   if (final.statistic < initial.statistic)
     {
        variable pars = get_par ([1,2,3]);
        @x0_ref = f.x0 + pars[0];
        @y0_ref = f.y0 + pars[1];
        @a_ref = pars[2];
        return 1;
     }

   return 0;
}

%}}}

define mho_find_source_extent (events, x0, y0, crude_src_size, img_size, bin_factor)
{
   variable adjustable_bin_factor = 0;

   if (crude_src_size <= 0)
     {
        throw ApplicationError, "crude_src_size > 0 is required";
     }

   if (img_size <= 0)
     {
        % img_size should be _much_ larger than
        % the max allowed elliptical region size
        variable max_major_axis = 2 * (Mho_Max_Size_Factor * crude_src_size);
        img_size = 10 * max_major_axis;
     }

   if (bin_factor <= 0)
     {
        adjustable_bin_factor = 1;
        if (img_size < 128)
          bin_factor = 1;
```

```
        else
          bin_factor = nint(img_size/128);
     }

   variable f = extract_image (events, x0, y0, img_size, img_size, bin_factor);

   if (adjustable_bin_factor && (bin_factor > 1))
     {
        % If the source region looks bright, don't use coarse binning.
        forever
          {
             variable dims = array_shape(f.image);
             variable ny = dims[0], nx=dims[1],
               iy = [ny/2-ny/8:ny/2+ny/8],
               ix = [nx/2-nx/8:nx/2+nx/8];
             if (max(f.image[iy,ix]) < 100 or bin_factor == 1)
               break;
             bin_factor /= 2;
             f = extract_image (events, x0, y0, img_size, img_size, bin_factor);
          }
     }

   variable a_guess;
   if (improve_source_position (f, crude_src_size, &x0, &y0, &a_guess))
     {
        f = extract_image (events, x0, y0, img_size, img_size, bin_factor);
     }

   variable pars = _mho_optimize_axes (f, a_guess, crude_src_size);

#iftrue
   % source = 2-D Gaussian => sigma_i = a_i / sqrt(3)
   pars[[0:1]] /= sqrt(3);
#else
   % source = uniform circular disk =>  R = a * sqrt(2)
   pars[[0:1]] *= sqrt(2);
#endif

   return [x0, y0, pars];
}
```