VODIA: VST-OmegaCAM Difference Image Analyis
(User manual)

Alfredo Volpicelli, Roberto Silvotti, Silvio Leccia, Fedor Getman, Juan Alcalà
INAF – Osservatorio Astronomico di Capodimonte, Napoli, Italy

March 28, 2008

Contents

Chapter 1

Introduction

Image subtraction is a method by which one image is matched against another by using a convolution kernel, so that they can be differenced to detect and measure variable objects.

Although this method requires much more computing time respect to the methods that work in the catalog space, it is of more general use and in particular it is more robust in relatively crowded fields.

Instead of writing a new code from scratch, we looked into the literature and we decided to adopt the software **DIA** (**D**ifference **I**mage **A**nalysis), written by Woźniak (2000). This package is very similar to the **ISIS** code written by Alard (2000): both are based on the "optimal PSF matching algorithm" with a space-varying kernel (Alard & Lupton 1998, Alard 2000) and were extensively used by different groups. In particular, the DIA software was used by the OGLE team (see e.g. Woźniak 2002) and therefore is optimized to detect small photometric variations in crowded fields.

DIA was then modified in order to increase the efficiency (parallelization) and match the requirements of large arrays of CCDs. Moreover new features were introduced in order to obtain a more flexible and more user friendly software. The result is a modified version called as **VODIA** (**V**ST **O**megaCAM **D**ifference **I**mage **A**nalysis), that includes the following improvements:

1. **Masks**: masks have been introduced for each scientific frame in order to exclude bad pixels.

2. **WCS**: a new tool makes use of the astrometric solution for the alignement of the scientific images. The astrometric information (CRVAL,CRPIX,CD) is read from the headers and, through the WCS library, the common part of the images is found and the images are cutted accordingly.

3. **External domains**: it is now possible to use an external list of "good stars" from which to extract the PSF. This feature is usefull for particular fields (e.g. very crowded or very empty fields), where the selection of the stars can be critical. Previously, the only possibility was to use an automated algorithm for the selection of "good stars". Moreover PSF images are now created and saved in fits format so that one can verify the quality of the PSF extraction.

4. **Photometry**: the output has been modified in order to have real fluxes (i.e. flux of each difference images plus flux of the reference image) in the light curves. Previously only the fluxes of the difference images were given. This gives the possibility to obtain also the relative amplitudes. More details on this item are given in section 2.7.

5. **Read-out noise**: the error due to the CCD reading has been introduced and summed (quadratically) to the shot noise.

6. **Ascii catalogues**: the output catalogues are now given in ascii format (previously they were binary files), according with the AW source list class: for each variable candidate VODIA gives RA and DEC (previously only x and y were given) and the series of measurements vs. time (light curve). The time of each frame is now taken automatically from the header, and the central time of each exposure (in JD) is calculated and reported in the light curve. Moreover barycentric and heliocentric time corrections are also given.

7. **Parallelization and python interface**: the program can work on a BEOWULF cluster with a simple parallel scheme (default is 1 CCD per node). The python interface, builded following AW standards, allows to retrieve the input images from the Data Base (DB) and save in DB all the configuration files; the output catalogue is saved in the data storage. More details on the python interface are given in chapter 3.

The potential scientific applications of this package are quite large. All the programs based on image subtraction are particularly good when used in crowded fields. But the advantage of using a spatially variable kernel is that it becomes possible to enlarge significantly the dimensions of the subwindows without loosing performance. This because the spatial variability of the kernel will compensate for the PSF distorsions across the field. Therefore VODIA can be used also in empty fields provided that the subwindows are large enough to contain several "good stars" from which derive the PSF. Indeed, in the case of the VST/OmegaCAM images, each subwindow will correspond to a CCD and therefore will contain more than enough "good stars" even when short exposure times are used.

The VODIA package consists of six independent *c* programs (see figure 1 for an overview of the program) described in detail in chapter 2.

The software can be used either in the **Astro-Wise Environment** (AWE) through an ad hoc python interface, described in chapter 3 or can run in shell mode (see chapter 4).

VODIA has been tested extensively on different data sets. The results of these tests are summarized in chapter 5.

Finally, chapter 6 contains some practical suggestions and **F**requently **A**sked **Q**uestions (**FAQs**).

Lettering : *the names of the packages are written in* san serif, *while the names of the parameters are in brackets and in CAPITALS.*

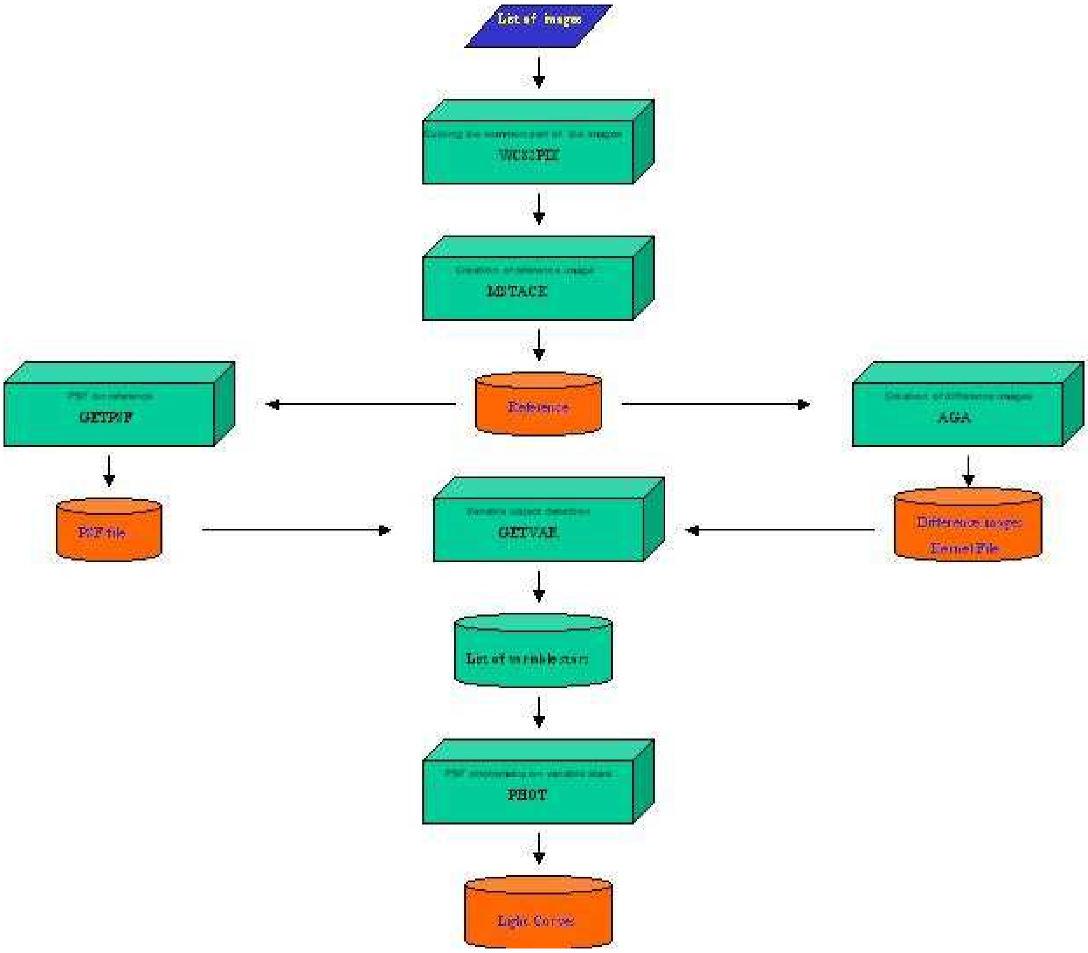


Figure 1.1: Vodia flowchart.

Chapter 2

Description of the c programs

2.1 Preparation of the images: wcs2pix

wcs2pix uses the astrometric solution for the alignement of the scientific images. The astrometric information is read from the headers and, through the WCS library, the common part of the images is found and the images are cutted accordingly.

2.2 Mstack: Image stacking

This step prepares the reference image. Mstack simply stacks a list of best seeing images and removes saturated pixels and bad pixels in the input images.

The software needs to be adjusted for different background and scaling of each frame. It starts with a series of frames and it uses the first of them as a reference for background and scaling. For each frame it prepares an histogram of pixel values (HIST_NBIN, HIST_LOW, HIST_HIGH). The histogram is dominated by a broad sky peak at low values followed by a much weaker wing due to the stars, which extends all the way to the saturation level. In a crowded field the sky peak is heavily skewed by faint stars. We found that the simple fit for the scaling and the background difference to all the pixels in the image is very unreliable.

For what concerns the background, the results are sufficiently accurate if we take the region inside the FWHM of the peak corresponding to the background and we consider the flux such that 30% (BKG_FRAC) of the pixels in this trimmed distribution lies below (see figure 2 for more details).

Then, concerning the scaling factor, we consider the ratio of the values of all the bright pixels in each image to the value in the first image (after having subtracted the background). This method to calculate the scaling factor requires that all the images used to build the reference frame have similar seeing. To assure that the pixels considered to derive the scaling factors belong to stars, only those having at least 300 (THRESHOLD) counts above the upper boundary of the FWHM region of the sky peak are used.

To produce the reference image, the user can choose between median or mean (MEDIAN).

2.2.1 The configuration file: mstack.par

```
NX0 = 2000          frame size in x
NY0 = 2000          frame size in y
NY = 2000           subframe size in y
```

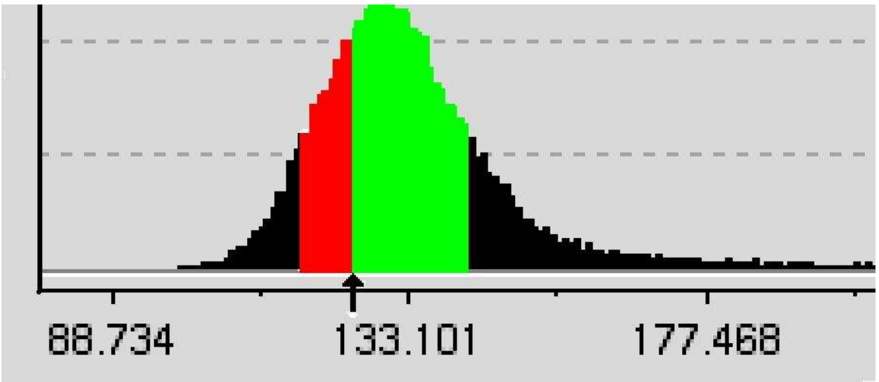


Figure 2.1: Background subtraction: the figure represents the number of pixels (y axis) vs. their value in a WFI scientific image. The black arrow indicates the value representative of the background, which divides the FWHM of the background peak into two parts: 30% of the pixels (left, red) and remaining 70% (right, green).

```
HIST_NBIN = 2000    n° histogram bins
HIST_LOW = 0.0      first histogram bin
HIST_HIGH = 2000.0  last histogram bin
NBIN_SMOOTH = 6     smoothing histogram bins
BAD_VALUE = 65000.0 bad pixel value (in output file)
MIN_LEVEL = 10      min value
SAT_LEVEL = 55000.0 max value
BKG_FRAC  = 0.3     background fraction
THRESHOLD = 300.0   threshold
MIN_SCALE = 0.5     min factor scale
MAX_SCALE = 2.0     max factor scale
                    (if the value for the scaling factor is not
                    included in the range [MIN_SCALE,MAX_SCALE]
                    the frame is rejected)
MIN_NGOOD = 3       min number of good pixels to obtain
                    the median or the average of that
                    pixel in the reference frame.
                    (otherwise it will be set to 65000)
VERBOSE = 1         show information on the screen (0=no info)
NSIG = 5.0          never used in the program!
MEDIAN = 1          (0=mean, 1=median)
```

2.3 Getpsf: PSF on the Reference

The PSF derived by Getpsf will not be used to obtain the matching kernel and the difference frames but only to extract PSF photometry from the difference images. Substantial part of the program Getpsf deals with the selection of good PSF stars.

First the full list of candidate objects is selected at local maxima of the intensity, for which the highest pixel stands out by more than 2 (NSIG_CLIP) sigmas above the background, where sigma is the photon noise estimate. A simplistic value of the flux is calculated using an aperture with 3 pixel radius (APRAD). The frame is divided into (NBOX_X $\times$ NBOX_Y) boxes and the 100 (NPSF_MAX) brightest stars from the bright end of the luminosity function are selected. The peak value for a star is refined using a parabolic fit in a 3×3 (FITRAD) pixel area around the central pixel. The ratio of the background subtracted peak to the total flux in the object (light concentration) is required to be less than 20% (RAT_THRESH) for each star in order to reject cosmic rays, for which this parameter is much larger. The sample of well behaved stars is cleaned of bad objects, e.g. cosmic rays and very tight blends, using sigma clipping on the distribution of light concentrations (NSIG_RAT). Finally, candidates for the fit are checked for close neighbors. The star is rejected from the fit if in an area of ± 3 (ISOHW) pixels around the peak there is another local maximum at least 2 (NSIG_DETECT) sigmas above the background and brighter than (0 (ISO_OFF) + 0.15 (ISO_SLO) $\times$ r) $\times$ fpeak, where r is the distance in pixels and fpeak is the peak flux of the star. With typical FWHM seeing values of about 3 pixels, this excludes stars that have fluxes significantly contaminated due to crowding.

The PSF model consists of 2 (NGAUSS) Gaussian, one for the core and one a factor of 1.83 wider for the wings, each multiplied by a 3rd order polynomial (NDEG_LOCAL). Both Gaussians are elliptical. The position angle of the major axis (PSF_AX,PSF_AY) and ellipticity (PSF_COS,PSF_SIN) can vary but remain the same for the core and wing components.

Spatial variability is modeled by allowing each of the local polynomial coefficients to be a function of X and Y. This function is again a polinomial of 2nd order (NDEG_SPAT). The first guess for the shape of the Gaussians is taken to be circularly symmetric and the initial FWHM of the core component is 3 pixels (FITRAD).

The linear and nonlinear parts of the fit are separated. Also the individual amplitudes of the stars must be taken out before fitting to generic PSF parameters. Therefore, in each iteration we fist solve the linear problem for all polynomial coefficients, then we update the shape of the Gaussian using moments of the light distribution of the current fit and finally we correct centroids of fitted objects with linearized least squares and recalculate the norm of each star. To avoid any potential instability the first few iterations are done with spatial variability turned off and then the full fit can be safely completed. In our tests only 2 (NITER) iterations were required at each stage.

PSF coefficients are stored in a binary file for later use (detection of variables on the reference frame and psf photometry on difference images).

2.3.1 The configuration file: getpsf.par

```
NBOX_X = 4           number box in x
NBOX_Y = 4           number box in y
NDEG_SPAT = 2        degree spatial variability
NDEG_LOCAL = 3       degree local solution
NGAUSS = 2           number of gaussian used
NPSF_MAX = 100       initial number of stars for fit
NOBJ_INIT = 128      min number objects to initialize
MIN_NBOX = 2         min number of stars in a box
MIN_FLUX = 3000.0    min value of star flux
MAX_THRESH = 500.0   (peak value - background) must be more
                     than this value
RAT_THRESH = 0.2     light concentration threshold
NSIG_DETECT = 2.0    if (peak - bg)*(peak - bg)*GAIN/peak >
                     NSIG_DETECT and background>0
                     then this is an object
CONTRAST = 2.0       check if (max-min)*(max-min)*gain
                     >= contrast*contrast*max)
```

```
where max and min are obtained in a
                        (2PEAKHW+1)*(2PEAKHW+1) pixels window.
    NSIG_RAT = 3.0      number of sigma for sigma clipping on
                        light concentration.
    PSFHW = 7           nunber of pixels (half window) for psf
    MAXHW = 1           check if the max pixel in a
                        (2MAXHW+1)*(2MAXH+1) pixels window
                        is the peak in a window of size
                        (2PEAKHW+1)*(2PEAKHW+1)
    PEAKHW = 4          peak half window
    ISOHW = 3           check in a window if the
                        pixel is isolated
    ISO_OFF = 0.0       offset neighbors
    ISO_SLO = 0.15      neighbors parameter
    FITRAD = 3.0        fitting radius
    APRAD = 3.0         aperture radius
    ANRAD1 = 10.0       background ring (internal radius)
    ANRAD2 = 15.0       background ring (external radius)
    BKG_FRAC = 0.3      all pixels in the background ring are sorted
                        and the background value is taken such that
                        30% of the pixel distribution lies below
(see Figure 2).
    NSIG_CLIP = 2.0     number of sigma for clipping
    NITER_INIT = 2      number of iterations for initial fitting
                        (constant PSF)
    NITER = 2           number of iterations for final fit
                        (variable PSF)
    VERBOSE = 1         show information on the screen
    RECENTER = 1        1  = ricalculate the center for psf
                        (0 = turned off)
    PSF_COS = 1.0       initial value
    PSF_SIN = 0.0       initial value
    PSF_AX = -0.8       initial value
    PSF_AY = -0.8       initial value
    SIGMA_INC = 0.548    sigma incrementation for gaussians
    SIGMA_MSCALE = 1.582 scaling of the semiaxis of the
                        elliptical gaussians
    GAIN = 2.0          gain
    SAT_LEVEL = 55000.0 saturation level
    MIN_LEVEL = 100.0   min level
```

2.4 **Aga:** Image subtraction

2.4.1 Basic equations

For the image subtraction method (Alard & Lupton 1998), the main goal is to find a convolution kernel K, that will trasform a reference image R to fit a given image I. In terms of least squares, this is equivalent to solve for the kernel that minimizes the sum:

$$\sum_i ([R \otimes K](x_i, y_i) - I(x_i, y_i))^2 \tag{2.1}$$

The kernel can be decomposed onto n basic Gaussian functions:

$$K(u,v) = \sum_n a_n K_n(u,v) = \sum_{ncomp} \sum_{d^x_{ncomp}} \sum_{d^y_{ncomp}} [a_{d^x_{ncomp}, d^y_{ncomp}} \, e^{\frac{-(u^2+v^2)}{2\sigma_{ncomp}}} u^{d^x_{ncomp}} v^{d^y_{ncomp}}] \tag{2.2}$$

where $\quad 0 < d^x_{ncomp} \leq D_{ncomp} \quad$ and $\quad 0 < d^x_{ncomp} + d^y_{ncomp} \leq D_{ncomp}$
D_{ncomp} being the degree of the polymonial corresponding to the n^{th} Gaussian component.
There are a total of $\frac{(D_{ncomp}+1)(D_{ncomp}+2)}{2}$ terms for each value of *ncomp*.

The kernel solution can be calculated by solving the linear system: $\quad Ma = B$
with

$$M_{ij} = \int [R \otimes K_i](x,y) \frac{[R \otimes K_j](x,y)}{\sigma(x,y)^2} dxdy$$

and

$$B_i = \int I(x,y) \frac{[R \otimes K_i](x,y)}{\sigma(x,y)^2} dxdy$$

2.4.2 Space varying kernel equations

The image subtraction can be generalized also at a spatial variability of the kernel (Alard 2000). In this case the kernel is built from the combination of coefficients spatially variable and a local constant kernel.

$$I(x,y) + Bkg(x,y) = Ker(x,y,u,v) \otimes Ref(x,y) \tag{2.3}$$

In this equation there is also the background component that is subtracted by the software after modelling with the polynomial expression:

$$Bkg(x,y) = \sum_i \sum_j a_i x^i y^j$$

with $\quad 0 < i \leq D^{bg} \quad$ and $\quad 0 < i + j \leq D^{bg}, \quad D^{bg}$ being the degree of the polynomial used to model the differential background variation (the default value for this parameter in aga.par is BKG_DEG = 1). The Kernel can be written in this way:

$$Ker(x,y,u,v) = \sum_n a_n(x,y) K_n(u,v) =$$

$$= \sum_{ncomp} \sum_{d^x_{ncomp}} \sum_{d^y_{ncomp}} \sum_{\delta^x} \sum_{\delta^y} [a_{\delta^x, \delta^y} x^{\delta^x} y^{\delta^y} e^{\frac{-(u^2+v^2)}{2\sigma_{ncomp}}} u^{d^x_{ncomp}} v^{d^y_{ncomp}}] \tag{2.4}$$

where $\quad 0 < d^x_{ncomp} \leq D_{ncomp} \quad$ and $\quad 0 < d^x_{ncomp} + d^y_{ncomp} \leq D_{ncomp}$
D_{ncomp} being the polynomial corresponding to the n^{th} Gaussian component.

These are N_COMP Gaussians modified by polynomials. The polynomial for the first component has degree DEG_1 and each subsequent one has a degree increased by DEG_INC (DEG_i+1 = DEG_i + DEG_INC). SIG_GAUSS_INC is the multiplicative increment for sigma of the Gaussian and the first Gaussian has sigma of SIG_GAUSS_1 (SIG_GAUSS_1 = 1/sqrt(2) and SIG_GAUSS_INC = sqrt(3) works fine).

The default values of these parameters are N_COMP=3, DEG_1=4, DEG_INC=–1, SIG_GAUSS_1=0.78 and SIG_GAUSS_INC=1.732 so D_0=4 with σ_0=0.78, D_1=3 with σ_1=1.35 and D_2=2 with σ_2=2.34 and the total number of terms for a constant kernel solution is 31.

The software performs the convolutions directly in real space, using a 15x15 pixel rasters for kernel components (the relative parameter is KER_HW that is the half width of the kernel; default value is 7).

Spatial variability is introduced by allowing each of the local kernel coefficient $a_n(x, y)$ to be a smooth function of x and y. The spatial variability depends on D^{spat} ($0 < \delta^x \leq D^{spat}$, $0 < \delta^x + \delta^y \leq D^{spat}$).
The default value for this parameter in aga.par is WDEG_SPATIAL = 2 and this means that for each local kernel there are 6 possible configurations.
The spatially variable problem quickly grows, the number of coefficients is

$$\frac{(D^{spat}+1)(D^{spat}+2)}{2}$$

times larger than in the case of a constant kernel; in fact the number of coefficients is given by:

$$n = \sum_{ncomp} \frac{(D_{ncomp}+1)(D_{ncomp}+2)}{2} \, \frac{(D^{spat}+1)(D^{spat}+2)}{2}$$

The first step is convolving a reference image with each piece of the kernel to form a set of images, which can be viewed as the basic vector:

$$V_n(x, y) = W_m(x, y) \; P_q(x, y)$$

where

$$W_m(x, y) = \frac{[R \otimes K_m](x, y)}{\sigma(x, y)}$$

and

$$P_q(x, y) = x^i y^j$$

Of course the index m depends on the number of local kernels

$$\sum_{ncomp} \frac{(D_{ncomp}+1)(D_{ncomp}+2)}{2}$$

and q depends from the spatial variability: $\frac{(D^{spat}+1)(D^{spat}+2)}{2}$.

Solving Equation 2.3 means finding a linear combination of these basic vectors that closest reproduces the light distribution of the frame.
Now it is possible to define the elements of the least squares matrix as scalar products of the least squares vectors:

$$M_{n1,n2} = \int W_{m1}(x,y)W_{m2}(x,y)P_{q1}(x,y)P_{q2}(x,y)dxdy \tag{2.5}$$

This integral extends over all the image. Since the computing time is crucial, it is possible to divide the integration domain into small rectangular sub-areas called square-domains.
The matrix elements can be written as:

$$M_{n1,n2} = \sum_k Q^k_{m1,m2}(x,y)P_{q1}(x_k,y_k)P_{q2}(x_k,y_k) \tag{2.6}$$

with the following definition for the integral in the rectangular domain Dm_k:

$$Q^k_{m1,m2} = \int_{Dm_k} W_{m1}(x,y)W_{m2}(x,y)dxdy$$

If we take these domains sufficiently small, the PSF variability can be ignored inside a single domain and we can assume that the kernel is constant. This corresponds to approximating x and y by the coordinates of the centers of the sub-areas, (x_k, y_k). The dimensions of the domains are defined by the parameter DOM_HW (the default value is 11 so the square domain is $(2 \times 11 + 1) \times (2 \times 11 + 1) = 23 \times 23$ pixels). Three options of domain patterns are available changing the parameter DOMAIN_MODE:

- uniformly distributed (value 0)
- centered on bright stars (value 1)
- centered on stars given by an external file with coordinates (value 2)

The number of square domains is determined by the parameters NDOM_X and NDOM_Y (the default is 20x10).

We see that the matrix elements corresponding to fitting the spatial variations of the kernel can be deduced from the matrix $Q^k_{m1,m2}$ for only the cost of the sum for all sub-areas.
The matrix $Q^k_{m1,m2}$ has a much smaller number of elements that the matrix $M^k_{n1,n2}$ and corresponds to the constant kernel solution:

$$Q^k_{m1,m2} = \int_{Dm_k} [R \otimes K_{m1}](x,y)\frac{[R \otimes K_{m2}](x,y)}{\sigma(x,y)^2}dxdy$$

Once $Q^k_{m1,m2}$ has been estimated, using the equation 2.6, it is possible to calculate $M_{n1,n2}$.

A similar method can be applied to expand the vector B:

$$B_n = \sum_k W^k_m(x,y)P_{q1}(x_k,y_k)$$

with:

$$W^k_m(x,y) = \int_{Dm_k} I(x,y)\frac{W_{m1}(x,y)}{\sigma(x,y)^2}dxdy$$

To find a solution for the equation 2.3, the software uses the LU decomposition[1] and starts with a initial solution that is cleaned through sigma clipping of individual pixels within domains and clipping of the entire distribution of whole domains using their χ^2 per pixel values. The software requires that after sigma clipping a fraction of at least MIN_AREA_DOM must be acceptable for each domain to enter the final solution (default value is 0.75, i.e. 75%). Also at least MIN_AREA (default 0.4 i.e 40%) of the total fitted area must be left in the fit and the χ^2 per pixel must be less than MAX_CHI2 (by default 8) to obtain a successful subtraction; otherwise the frame is rejected.

Masking of pixels is very important because even few bad pixels can disturb the solution. It may be worth to mask out very bright stars too. This is controlled by MIN_LEVEL (minimum pixel value to be used, must be positive!) and SAT_LEVEL (only pixels lower than that this value will be used). When a pixel outside this range is found all the pixels belonging to a window (2BAD_GROWRAD1+1)*(2BAD_GROWRAD1+1) centered on the bad pixel are masked; these pixels will be set to BAD_VALUE in output. It is strongly recommended to use for BAD_GROWRAD1 the same value as KER_HW.

BAD_VALUE is also used for all the pixels for which there is no solution due to an insufficient number of pixels left for the fit or when final χ^2 was too high.

MIN_AREA is the minimum fraction of the pixels in a given section that must be left after sigma clipping, otherwise the entire section is marked with BAD_VALUE. Also χ^2 must be lower than MAX_CHI2 or the section is marked with BAD_VALUE.

Sigma clipping is used to reject variables from the fit.

Maximum number of iterations is MAX_NITER and sigma cutting depends on N_SIG.

The parameter VERBOSE = 1 shows some information on the screen.

Finally, for each image, the kernel solution is written into file nameimage.ker; this file contains the kernel of each sub-section (there are NX_0/NX$\times$ NY_0/NY s ubsections for each image).

Pixels equal to 0 in the input "mask_image" are not used in the fit but will have normal value in the output image unless rejected otherwise.

In this version the software requires a mask for the reference image and a mask for each input frame. In fact the file that contains the list of images contains also the name of the masks.

Subtracted images in output will have a prefix "s_". Kernels have the extension ".ker". Kernel images are not normalized. Norms of the best fit kernels are written to the screen in VERBOSE mode.

Summary of the different steps of the algorithm:

1. The reference image is convolved with each of the basic kernel functions. These form the vectors with which we can fit any other image.

2. Least squares matrix is calculated once from the above vectors using $\sqrt{reference\_image}$ as an estimate of the noise.

3. Least squares matrix is cleaned according to the masking of the current image and sigma clipping starts. First iteration is just the fit and calculation of $\chi 2$. Each subsequent iteration subtracts from the least square equation contributions corresponding to rejected pixels and recalculates sigma.

4. At the end a given section of the image is checked for goodness of the solution according to MAX_CHI2 and MIN_AREA.

[1]The LU decomposition enables to write a generic matrix A as a product of two matrix the *lower triangular* **L** (that has non-zero elements only on the diagonal and below) and the *upper triangular* **U** (that has non-zero elements only on the diagonal and above). With this method we are able to solve a system of equations splitting it in two simplier systems.

2.4.3 The configuration file: aga.par

```
NX_0 = 2000             size frame in x
NY_0 = 2000             size frame in y
NX   = 500              size subframe in x (frame is divided
                        in subframes of NXxNY pixels).
        If you don't want subframes
                        NX must be equal to NX_0 - 2*Ker_HW
NY   = 500              size subframe in y
                        If you don't want subframes
                        NY must be equal to NY_0 - 2*Ker_HW
KER_HW = 7              kernel half window
N_COMP = 3              number of components
DEG_1 = 4               degree first component
SIG_GAUSS_1 = 0.78      sigma first gaussian
DEG_INC = -1            degree increment
SIG_GAUSS_INC = 1.732   sigma gaussian increment
BKG_DEG = 1             degree background
GAIN = 2.0              gain (electrons/ADU)
MIN_LEVEL = 100.0       min value
SAT_LEVEL = 55000.0     saturation value
BAD_VALUE = 65000       bad value
BAD_GROWRAD1 = 7        half window for masking
BAD_GROWRAD2 = 2        half window for masking (not used)
MIN_AREA = 0.4          mininum good total area for fit
MAX_NITER = 1           max number of iteration for clipping
N_SIG = 3.0             n sigma for clipping
MAX_CHI2 = 8.0          max chi2
VERBOSE = 3             show information on the screen (3=more info)
WDEG_SPATIAL = 2        degree spatial variability
DOM_HW = 11             domain half window
NDOM_X = 20             number domains in x
NDOM_Y = 10             number domains in y
DOM_THRESH = 50.0       domain threshold
N_SIG_DOM = 3.0         n sigma for domains
DOMAIN_MODE = 1         0 = domains uniformly distributed
                            over the image
                        1 = domains around bright stars
                        2 = domains around stars
                            from an external file*
MOHW = 3                half window to find the maximum within
                        a domain
N_ITER_DOM = 1          n iterations clip domains
MIN_AREA_DOM = 0.75     min good area domain
MIN_NKEEP = 50          min number of domains

*You have to create an ascii file named coordinate.txt in which
 you write x and y of each ``good star'' (one star per line).
```

2.5 Getvar: find variable candidates

This programme detects variable objects using some preliminary variability measurements based on the entire series of difference images, creates the list of variable candidates and makes final measurements for these candidates on the reference image.

This way our database will contain only the light curves of the variable candidates, whereas nonvariable stars will not be included.

Program getvar starts by rejecting some fraction of the frames with the worst seeing, in our case 10% (FWHM_FRAC = 0.9).

In the next step we consider individual light curves in the 3×3 (2FILTER_HW+1 ×2FILTER_HW+1) pixels square apertures centered on every pixel. This corresponds to a smoothing of all the difference frames with the 3 pixels wide mean filter before examining pixel light curves. Some points are rejected because of saturated (SAT_LEVEL) and dead pixels (ERR_CODE = 65000) and we require that the cleaned light curve contains at least 50% of the measurements.

For each pixel light curve we take the median flux to be the base line flux and analyze the ratios of the departures from this base level to their noise estimates.

The pixel is declared as variable if one of the following two conditions are met:

- "Continuous" variables: there are at least 3 (NCONS_VAR1) consecutive points departing at least 3σ (NSIG_VAR1) from the base line in the same direction (up or down);

  or

- "Transient" variables: there are at least 10 (NPTS_VAR2) points in total (not necessarily consecutive) departing at least 4σ (NSIG_VAR2) from the base line in the same direction.

In the next step we label variable pixels according to the ratio of the number of the deviating points from the above cut which depart upwards to the number of points departing downwards.

If the ratio is between 0.5 (1/LIM_RATIO) and 2.0 (LIM_RATIO) we fill a corresponding pixel of the variability image for "continuous" variables.

Otherwise we fill the pixel of the image for "transients". As the measure of pixel variability we adopt $D = \sum_i |F_i - F_0|$ where F_i is the flux and F_0 is the base line flux.

For "continuous" variables the pixel value in the variability image (CONTINUOUS_VAR.fits) will be: $(D_{up} + D_{down})/(n_{up} + n_{down})$ where n is the number of points high or low with respect to F_0. For "transient" variables the pixel value in the variability image (TRANSIENT_VAR.fits) will be: D_{up}/n_{up} or D_{down}/n_{down} depending whether $n_{up} >= n_{down}$ or $n_{up} < n_{down}$ respectively.

After the variability images are builted, we can look for groups of variable pixels.

With the above definition of D sufficiently high signal to noise variables will produce groups of pixels resembling the local PSF shape. Therefore the last step is to detect point sources in variability images using a PSF model, a very similar procedure to our star detection algorithm. Just like in the case of star detection, we look for local maxima of the correlation coefficient with the PSF in the excess of 0.8 (C_MIN). In the end we have two lists of candidate variables of the basic types described.

Then the centroid of each variable is determined using the entire series of difference images. The program takes 9×9 (2CENTER_HW+1 × 2CENTER_HW+1) pixel rasters centered on each variable using only the frames for which difference between the measured flux and the template flux is at least 3σ (NCONS_VAR1). The absolute values of these rasters are subsequently weighted by their signal to noise and coadded to accumulate as much signal as possible in the peak.

Using a 3×3 neighborhood of this peak and a parabolic fit of the peak itself, we calculate the centroid. Cross identification of variables from "continuous" and "transient" channels is done because some of them can in principle appear in both lists; variables closer than 2 (ID_RAD) pixels are treated as one and are given the value of variability type 11. Transients are type 1 and continuous variables are type 10.

We also set a crowding flag equal to 1 if there is a pixel brighter than (ISO_SLO) $0.15 \times r \times f$ in the ± 4 pixels (ISOHW) neighborhood, where r is the distance (in pixels) and f is the flux of the central pixel of the variable on the reference image.

Program getvar writes an ascii catalog (the name is taken in input) entry for each of the variables with this format:

1. X template coordinate
2. Y template coordinate
3. flux PSF photometry (reference image)
4. error PSF photometry (reference image)
5. flux aperture photometry (reference image)
6. error aperture photometry (reference image)
7. background
8. χ^2 per pixel of the PSF fit
9. correlation coefficient with the PSF
10. number of bad pixels
11. variability type (1="continuous" variable, 10="transient", 11=both)
12. number of frames used for centroid calculation
13. crowding flag (0=no crowding)

2.5.1 The Configuration file: getvar.par

```
APRAD = 3.0            aperture radius (used only to search
                       for variables in variability images
                       and for centroid determination in
                       difference images)
FITRAD = 3.0           radius for aperture/PSF photometry
NORMRAD = 6.0          radius for psf model
ANRAD1 = 7.0           internal radius of background ring
ANRAD2 = 12.0          external radius of background ring
BKG_MODE = 1           1 = background subtraction
                       0 = keep background
NSIG_BKG = 2.0         n sigma background
SAT_LEVEL = 55000.0    saturation level
MIN_LEVEL = 10.0       min level
BAD_VALUE = -99.0      bad value
GAIN = 2.0             gain (electrons/ADU)
EXPTIME = 500.0        exposure time (sec)
ERR_CODE = 65000.0     pixel error code
VERBOSE = 3            show information on the screen (3=more info)
```

```
SMHW = 1                 Search Maximum Half Window
                         (to detect stars)
MOHW = 2                 Convolves image im(nx,ny) with the
                         PSF model (2*MOHW+1)x(2*MOHW+1).
C_MIN = 0.85             THRESHOLD TO DETECT VARIABLE CANDIDATES
NOBJ_INIT = 64           initial number of objects detected
ID_RAD = 2.0             minimum distance for cross
                         correlation
NCONS_VAR1 = 3           n° consecutive points continuous var
NPTS_VAR2 = 10           n° points transient var
NSIG_VAR1 = 3.0          n sigma continuous var
NSIG_VAR2 = 4.0          n sigma transient var
LIM_RATIO = 2.0          ratio (see the text)
ISOHW = 4                half window to check for close
                         companions
ISO_OFF = 0.0            offset close compainions
ISO_SLO = 0.15           crowding parameter (see the text)
BAD_MARGIN = 14.0        bad margin
CENTER_HW = 4            half window to find object center
FILTER_HW = 1            half window for smoothing (see the
                         text)
FWHM_FRAC = 0.9          minimum rate of good frames
TMP_FILE = var.coo       ascii file that will be used by Phot
                         containing the variable candidates
                         (8 columns: x,y,RA,DEC,
                         PSF phot,err,AP phot,err)
```

2.6 PhotRef: photometry on the reference image

This program can be used instead of Getvar, only for specific applications when we want to obtain photometry of an external list of objects. This can be usefull, for example, when the user wants to obtain the light curve of all the objects in the field (variables + stable sources), in order to apply specific software e.g. to detect exoplanet transits. The configuration file and the usage are the same as in Getvar.

2.7 Phot: Light Curve

Program Phot receives in input a series of difference frames, a series of original frames (for noise estimates), coefficients of spatially variable PSF matching kernel for each subframe (files .ker) and finally coefficients of the PSF (psf file) for the reference frame.

By convolving local kernel and reference PSF for each subframe, it constructs a PSF profile at the position of each variable and calculates amplitudes a_{ap} and a_{psf} (normalised to 1 sec) with corresponding errors σ_{ap} and σ_{psf}, which involves little more than simple sums over pixels (see below).

We used $r_{ap} = r_{fit} = 3$ (FITRAD) pixels. Whenever there is no information, phot will put requested error codes to keep the record of such gaps.

$$a_{ap} = \sum_{i}^{r_i < r_{ap}} f_i \tag{2.7}$$

$$\sigma_{ap} = \sqrt{\sum_i \sigma_i^2} \tag{2.8}$$

$$\sigma_i^2 = \frac{f_{i,0}}{GAIN} + \frac{RON^2}{GAIN^2} \tag{2.9}$$

$$a_{psf} = \frac{\sum_i (f_i P_i / \sigma_i^2)}{\sum_i (P_i^2 / \sigma_i^2)} \tag{2.10}$$

$$\sigma_{psf} = \frac{1}{\sqrt{\sum_i (P_i^2 / \sigma_i^2)}} \tag{2.11}$$

$$\chi^2 = \sum_i^{r_i < r_{fit}} \frac{(a_{psf} P_i - f_i)^2}{\sigma_i^2} \tag{2.12}$$

where f_i is the difference flux at pixel i, $f_{i,0}$ is the flux including background of the corresponding original image before subtraction, RON is the read-out-noise and P_i is the PSF profile at pixel i.

The final light curves are stored in a 22 columns single ascii file called as LCall.txt, which contains all the measurements of each variable candidate. The format of the ascii record is the following:

1. name of the original image
2. MJD
3. BJD (MJD corrected for solar system barycenter)
4. HJD (MJD corrected for solar barycenter)
5. X coordinate of object
6. Y coordinate of object
7. RA
8. DEC
9. flux PSF phot. on the Reference + flux PSF phot. on the difference image
10. error PSF flux on the Reference + error PSF flux on the diff. image
11. flux aperture phot. on the Reference + flux ap. phot. on the diff. image
12. error ap. phot. on the Reference + error ap. phot. on the diff. image
13. flux PSF photometry on the diff. image
14. error PSF photometry on the diff. image
15. flux aperture photometry on the diff. image
16. error aperture photometry on the diff. image
17. background

18. χ^2 of the PSF fit (relative to each object)

19. correlation coefficient with the PSF

20. χ^2 per pixel of kernel solution (relative to each frame, see Section 2.4)

21. FWHM of the PSF profile

22. number of bad pixels within the fitting radius

2.7.1 The configuration file: phot.par

```
APRAD = 4.0             not used
FITRAD = 3.0            radius for aperture/PSF photometry
NORMRAD = 6.0           radius for psf model
ANRAD1 = 6.0            internal radius of background ring
ANRAD2 = 7.0            external radius of background ring

BKG_MODE = 1            1 = background subtraction
                        0 = keep background
NSIG_BKG = 2.0          n sigma background
SAT_LEVEL = 55000.0     saturation level
MIN_LEVEL = 10.0        min level
BAD_VALUE = -99.0       bad value
GAIN = 2.0              gain (electrons/ADU)
EXPTIME = 500.0         exposure time (sec)
ERR_CODE = 65000.0      error code
VERBOSE = 1             show information on the screen
READ_NOISE = 0.         read-out-noise (electrons)
```

Chapter 3

The AW phyton interface

VODIA is fully compatible with the Astro-Wise Environment (AWE). This translates into the following characteristics:

- VODIA is able to run automatically launching a single python interface that calls all the c programs. The python interface creates all the 5 configuration files (one for each *c* program apart from wcs2pix) and send the processes to a BEOWULF cluster. The parallel scheme is the same used in the AW pipeline: each CCD corresponds to a single subwindow which is processed by a single node of the cluster.

  When running in automatic mode, the user can give a list of images for the creation of the reference image. If this list is not given, the reference is created from the coaddition of *all* the input frames. VODIA can not work on images having pixel with negative values, so the interface checks if the regridded frames are sky subtracted (looking at the swarp configuration file in the DB); if the images are skysubtracted VODIA is stopped and an error message appears on the screen. If the images are not skysubtracted but have some negative pixels, the interface replaces these pixels with a flag value in order to exclude them during the analysis. A warning message is printed on the screen.

- The python interface interacts with the Data Base (DB) in order to perform the following actions: retrieve the input files (images and masks), save into the DB all the configuration files, save in the data storage the output catalogue containing the variable candidates.

  To save all the configuration files means the complete reproducibility of the results, which is one of the key concepts of the AW system.

3.1 Inspect method

An inspect method was implemented in order to have a quicklook of the results.
For each variable candidate the inspect method performs the following tasks:

1. Plots the light curve and produces a stamp of the object.

2. Calculates and plots the amplitude spectrum (square root of the power spectrum) using a Discrete Fourier Transform.

3. Extracts from the amplitude spectrum the frequency with highest amplitude, calculates and plots the phase diagram (see figure 3.1).

A file containing all this information is produced for each variable candidate.

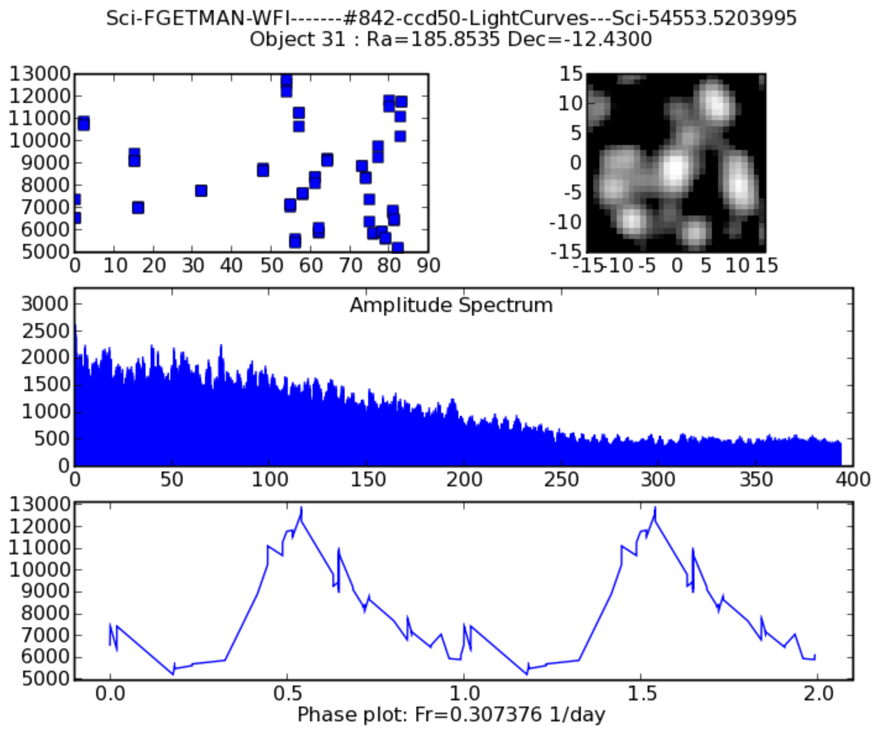


Figure 3.1: Plots created by the inspect method. Upper panels: light curve and stamp of the variable candidate. Middle panel: power spectrum. Lower panels: phase diagrams corresponding respectively to the frequency with highest amplitude and to a frequency given by the user (right).

3.2 An example of using VODIA in the AWE

The exercise that we describe here was proposed for the first time to the participants of the AW workshop that was held in Leiden in November 2005.

This exercise consists in analysing 68 500×500 pix images of the center of a globular cluster in order to detect variable stars. The images are already BIAS and FF corrected and aligned (astrometric solution already calculated). These images can be retrieved from AW data storage.

1. Once having entered AWE, this is the list of python commands:

```
[import all python scripts:]

awe> from astro.main.VariabilityFrame import VariabilityFrame
awe> from astro.external import Dia
awe> from astro.main.Config import DiaMstackConfig,DiaGetpsfConfig,
     DiaAgaConfig,DiaGetvarConfig,DiaPhotConfig

[select science frames:]
```

```
awe> q = RegriddedFrame.OBJECT.like('vartest')
awe> regs = [reg for reg in q]

[creates variability object:]

awe> v = VariabilityFrame()

[put our list of images as input:]

awe> v.regridded_frames = regs

[run VODIA (all programs automatically):]

awe> v.make()
```

2. Now you can use the inspect method to see the results:

   ```
   awe>v.inspect()
   ```

   You can save the output file with the results:

   ```
   awe>v.store()
   ```

   And finally you can store the results into DB:

   ```
   awe>v.commit()
   ```

3. After having looked at the light curves obtained with default parameters, one can try to play with the parameters (trying to optimize the results). We suggest to change only a few parameters in **getvar**, e.g. the detection threshold (C_MIN), that is presently set to 0.85, and/or the criteria to select the variables (NCONS_VAR1, NPTS_VAR2, NSIG_VAR1, NSIG_VAR2), which are presently set to 3, 10, 3.0 and 4.0 respectively.

   Note that now it is not necessary to run again the whole package but only **getvar** and **phot**. An example of changing one parameter and running again **getvar** and **phot** is given below.

   ```
   Example of changing C_MIN and running again getvar and phot:

   awe> v.getvar_config.C_MIN
   awe> v.getvar_config.C_MIN = 0.9
   awe> v.make_getvar()
   awe> v.make_phot()
   ```

Chapter 4

Running VODIA in shell mode

4.1 Running wcs2pix

- **Input**: list of the images to cut.
- **Output**: images cutted (with suffix sub).

```
In the path where there are the images:

> wcs2pix  *.fits
```

4.2 Running Mstack

To obtain the reference frame from the stacking of the best images, you need to create an ascii file with the list of file names (image_names_file).

- **Input**: parameter file, list of images (ascii file with a name of image per line)
- **Output**: reference image

```
usage: mstack <parameter_file> <image_names_file> <output_image>

example:

>>mstack mstack.par listmstackB.txt REFERENCE.fits

image: interp_B01.fits  bkg:   130.5  scale:  1.0000
image: interp_B02.fits  bkg:   130.5  scale:  0.9983
image: interp_B03.fits  bkg:   130.5  scale:  0.9950
image: interp_B04.fits  bkg:   181.5  scale:  0.9720
image: interp_B05.fits  bkg:   185.5  scale:  0.9716
image: interp_B06.fits  bkg:   187.5  scale:  0.9731
image: interp_B07.fits  rejected
image: interp_B08.fits  rejected
image: interp_B09.fits  rejected
image: interp_B10.fits  bkg:   354.5  scale:  0.9553
.......
```

```
.......
image: interp_B69.fits  bkg:   134.5  scale:  0.9414

image stack written to REFERENCE.fits
>>
```

4.3 Running **Getpsf**

- **Input**: parameter file, reference image.
- **Output**: file with psf information.

```
usage: getpsf <parameter_file> <input_image> <output_file>

Example:

>>getpsf getpsf.par REFERENCE.fits PSF.bin

Found 16189 good local maxima Found 99 psf stars

Initial fit (constant PSF):

iter    ax      ay      cos(fi) sin(fi)
 1      -0.355  -0.371  0.000   1.000
 2      -0.243  -0.257  0.005   1.000

Final fit (variable PSF):

iter         N(good)    N(clip)
 1             96          3
 2             95          4


PSF coefficients written to PSF.bin

>>
```

4.4 Running **Aga**

The Aga step is the core of the Variability Pipeline and requires several files: aga.par (parameters), a mask frame for the reference (16 bit, 0=bad, 1=good), the reference itself and an ascii list of all the input files (image_names_file). This file must contain at each line an image.fits and its mask.fits[1].
It can be useful to save the standard output in a log file to keep track of the computation because some information like the kernel normalization factors are not saved otherwise.

- **Input**: parameter file, mask of reference image, reference image, list of images and corresponding mask.

[1]Even if the images have the same mask, you have to create a mask for every image with different name.

- **Output**: files of the kernel (with same name of the image but with .ker as extention) and subtracted images (named as s_image-name), log file (optional).

```
usage: aga <parameter_file> <mask_image> <reference_image>
<image_names_file> > LOG_FILE
```

Example[2]:

```
>> aga aga.par mask_REFERENCE.fits REFERENCE.fits agalist.txt

  *** Image subtraction software ***

         69 image(s) to process

Section [1,1] of image 1 : bszinterp_B01.fits

Local sigma clipping of domains:

 iteration     chi2/dof     N(clip)      N(fit)
0
     0          3.2517          0     65724
1
     1          2.5191        517     65207
2

Sigma clipping of domain distribution:

iteration   median chi2/dof     sigma    N(good)
     1          2.4093          0.1267      122

rejected domain   40:   x=169,  y=172,  chi2_n= 4.720,  status=1
rejected domain   49:   x=200,  y=268,  chi2_n= 4.398,  status=1
rejected domain  109:   x=422,  y=144,  chi2_n= 5.978,  status=2
rejected domain  112:   x=435,  y=347,  chi2_n= 4.080,  status=1

    ....
    ....
    ....

Section [3,3] of image 69 : bszinterp_B69.fits

Local sigma clipping of domains:

 iteration     chi2/dof     N(clip)      N(fit)
0
     0          3.0900          0     97300
1
     1          1.7217        681     96619
2
```

[2] chi2/dof = chi2*gain

```
Sigma clipping of domain distribution:

iteration   median chi2/dof     sigma    N(good)
     1          1.4014          0.6094      180

rejected domain    0:   x= 33,  y= 55,  chi2_n= 7.614,  status=1
rejected domain    3:   x= 41,  y=189,  chi2_n= 5.727,  status=1
rejected domain   10:   x= 44,  y= 57,  chi2_n= 7.066,  status=1
rejected domain   11:   x= 44,  y= 77,  chi2_n= 5.782,  status=1
rejected domain  137:   x=383,  y=236,  chi2_n= 6.734,  status=1


% good_area  0.994227
% chi2_n     1.401430
% nkeep      180

Kernel norms:

bszinterp_B01.ker     1.01456
 .... ....

bszinterp_B67.ker     1.00526
bszinterp_B68.ker     1.00389
bszinterp_B69.ker     1.00418


>>
```

Status means why the domain was rejected:
status = 1 means sigma domain > N_SIG_DOM × sigma
status = 2 means area < MIN_AREA_DOM

4.4.1 Running Getvar

For this step we need only to prepare an ascii file with the list of files in this way:

```
subtracted_image.fits original_image.fits kernel.ker

Example:

s_interp_B01.fits interp_B01.fits interp_B01.ker
s_interp_B02.fits interp_B02.fits interp_B02.ker
s_interp_B03.fits interp_B03.fits interp_B03.ker
.......
```

Now it is posssible to run the command getvar:

- **Input**: parameter file, reference image, psf file (output of getpsf), list of image (build as described above), offset in x and y.

- **Output**: catalogue of variable objects.

```
usage: getvar <parameter_file> <ref_image> <psf_fit_file>
<image_data_list> <variable_catalog> <x0_off> <y0_off>

x0_off = add this value at final coordinates in x
y0_off = add this value at final coordinates in y


Example:

>>getvar getvar.par REFERENCE.fits PSF.bin listgetvar.txt B753.db 0 0

*** Variability finder for image subtraction ***

60 images

Found 8 and 1 candidates for variables of both types

var1:   0    42.869  1267.957    2313.0     1
var1:   1    43.928   522.882     657.4     1
var1:   2    56.859  1263.611    1765.7     1
var1:   3    63.472   380.416   12000.9     1
var1:   4   117.131  1411.253   10376.9     0
var1:   5   143.105  1008.252    3591.8     0
var1:   6   146.140   568.042     519.2     1
var1:   7   168.993   172.493    9198.2     1

var2:   0   238.616   375.000      65.2     0

Done !

The numbers on the screen represent respectively: variable type,
number, x, y, PSF photometry on the reference image and crowding
flag. The ascii catalog is in B753.db.
```

4.5 Running **Phot**

- **Input**: parameter file, reference image, psf file (getpsf output), images list, catalogue of variable objects (getvar output).
- **Output**: ascii file with light curves.

```
usage: phot <parameter_file> <ref_image> <psf_fit_file>
<image_data_list> <var_coords_file> <phot_database>


Example:
```

```
phot phot.par REFERENCE.fits PSF.bin listphot.txt var.coo LC.db

*** Profile photometry with variable PSF and kernel ***


69 images to process
8 variables to measure

image: 0 star: 0 flux:    -11.88 err:      1.52   nbad: 0
image: 0 star: 1 flux:    206.28 err:      8.34   nbad: 0
image: 0 star: 2 flux:     -6.47 err:      0.74   nbad: 0
image: 0 star: 3 flux:    -33.03 err:      4.33   nbad: 0
image: 0 star: 4 flux:    -33.03 err:      4.33   nbad: 0
image: 0 star: 5 flux:    335.96 err:      6.46   nbad: 0
image: 0 star: 6 flux:    199.07 err:      6.59   nbad: 0
image: 0 star: 7 flux:     76.76 err:      6.26   nbad: 0
image: 1 star: 0 flux:     -8.47 err:      1.56   nbad: 0
image: 1 star: 1 flux:    293.80 err:      8.43   nbad: 0
image: 1 star: 2 flux:     -4.09 err:      0.75   nbad: 0
image: 1 star: 3 flux:     -4.89 err:      4.39   nbad: 0
image: 1 star: 4 flux:    349.97 err:      6.61   nbad: 0
.....................

Appending photometry to:  LC.db

Done !
```

Note: the flux and relative error on the screen are from PSF photometry.

Chapter 5

Tests

VODIA was tested using four independent sets of images:

- 69 VLT FORS 2k×2k B band images of the globular cluster NGC 1866 (Musella et al. in preparation, see Fig. 3);
- 46 8k×8k WFI images of the Carina dwarf spheroidal galaxy in the B band (D'Allora et al. 2003);
- 15 WFI images of the Capodimonte Deep Field (OACDF, Alcalà et al. 2004) in the V band.
- 1063 1k×1k simulated images that contain 32 variable stars and 32 planet transits.

In the next subsections we will concentrate on the first (NGC 1866) and last (simulated images) data sets as these are the most significants tests, where we obtained quantitative results. However, it is worth to note that also in the other two experiments we were able to detect many variable stars with quite satisfactory results (at least qualitatively).

5.1 NGC 1866

These 69 FORS images span over 78 days and were analysed using three independent methods. Two of them (VODIA and another software developped by Johannes Koppenhoefer at USM) are based on image subtraction. An example of a difference image of NGC 1866 produced by VODIA is shown in Fig. 4. The third method consists on differential photometry applied to the catalogs obtained using ALLFRAME (an evolution of DAOPHOT developped by Peter Stetson).

The comparison of the results is shown in Fig. 5, 6 and 7, where we have selected five variable stars with different characteristics. Two of them (Fig. 5) are relatively bright and have very large amplitudes; the two stars in Fig. 6 are fainter, have smaller amplitudes and are located near the center of the cluster. Finally, the last one in Fig. 7 is one of the faintest object detected. As one can see, the results of the three methods are in good agreement for all the stars considered.

5.2 Simulated images

The advantage of using simulated images instead of real ones is that we can directly evaluate the efficiency of the program used (in terms of fraction of variable objects detected) and the accuracy of the measured amplitudes. 1063 simulated images were created using a software written by Johannes Koppenhoefer (USM). The time distribution is the following: 3 blocks of 7 nights each, separated by 21 nights with no images (corresponding to 3 weeks of data in bright time, giving a total duration of about 62 days. Each night has about 50 images with a 8.6 min separation between one image to the next. The PSF varies between 1.5 and 4 pixels (with small variations within the same night and stronger variations from night to night). The background is simulated with a second order polynomial. Then 13200 stable stars were introduced with an exponential flux distribution. Finally 32 variable stars and 32 planet transits were added. The variable stars have periods in the range 1 to 10 days and amplitudes of 10 to 50%. The planets transits are characterised by ..$\leq$P/days$\leq$.. and $0.005 \leq \Delta F/F \leq 0.04$ (F being the flux).

An example of simulated image is shown in Fig. 8.
The comparison between the original light curves (i.e. those which were used to create the simulated images) and the light curves found by VODIA is shown in Fig. 9, 10 (variable stars) and 11, 12 (planets). As one can see from these figures, the relative amplitude from aperture photometry are in relatively good agreement with the true amplitudes (within $\sim$10% for the variable stars), although they are less precise for the transits because of the very small relative variations of these events, which are comparable with the level of the noise.

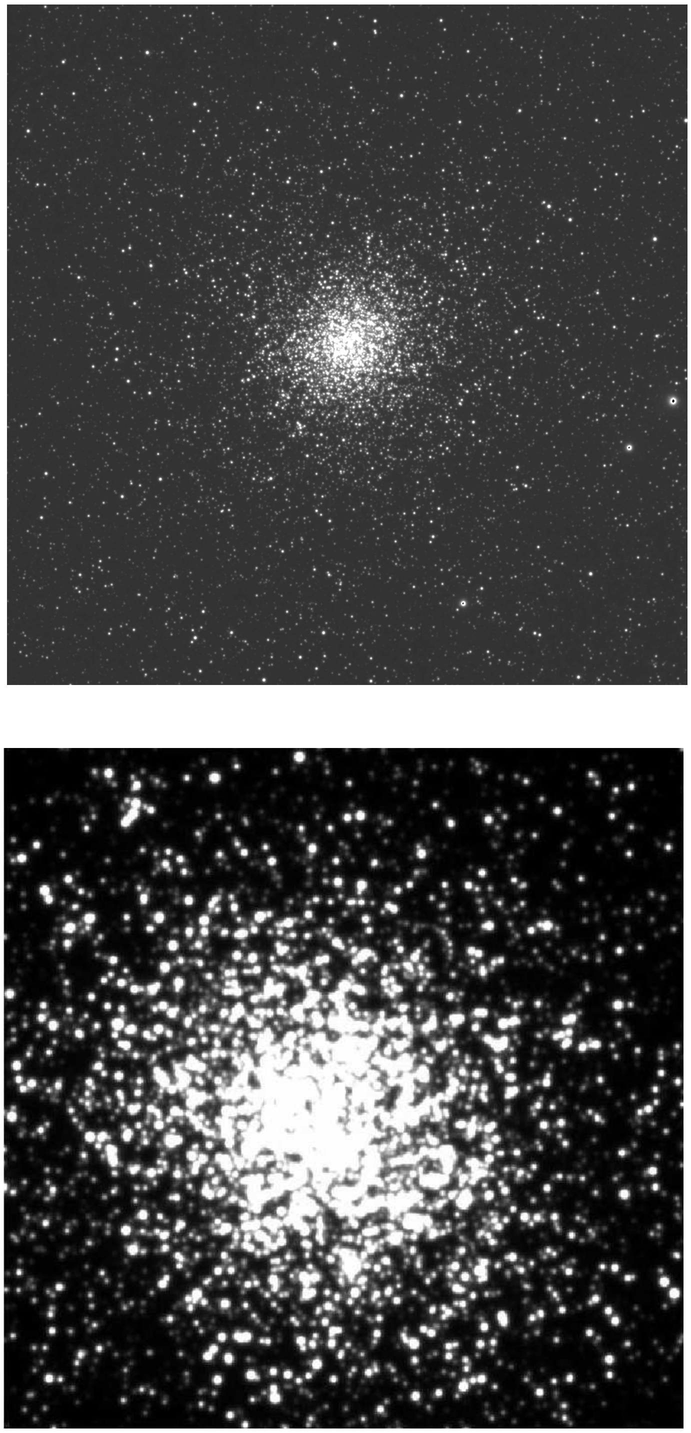


Figure 5.1: One of the FORS images of the globular cluster NGC 1866 in the LMC, which have been used for our first set of tests. The lower panel represents the center of the cluster.

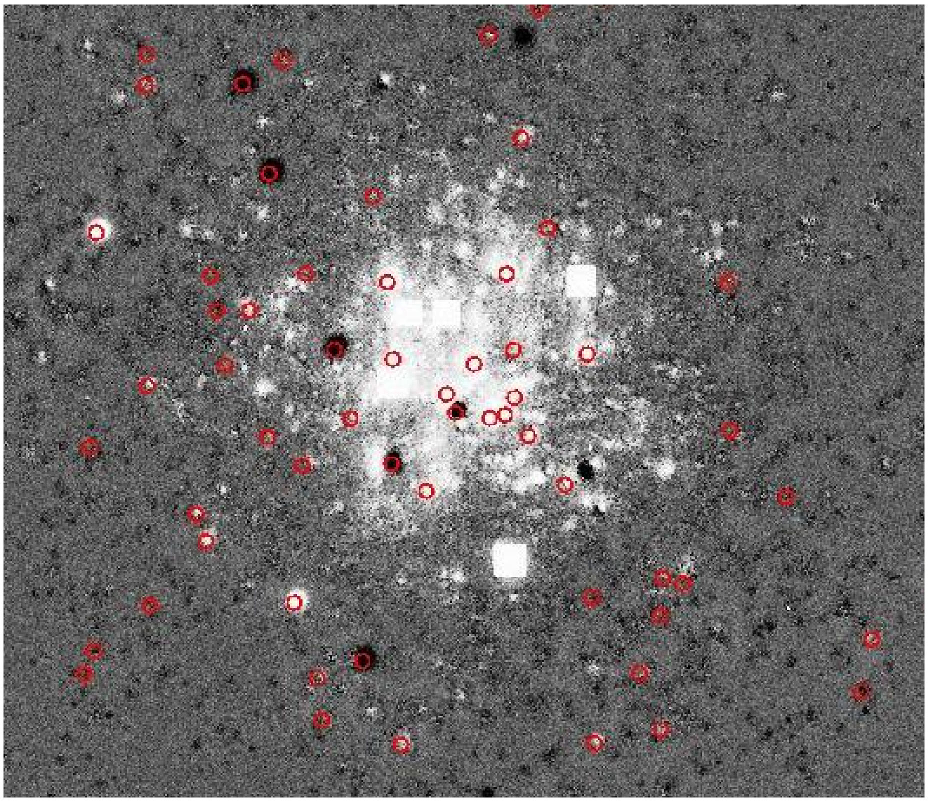


Figure 5.2: Example of a difference image obtained by VODIA: the variable candidates, represented by positive (white) or negative (black) objects, are outlined by red circles. The white squares are the masks that cover bad regions (e.g. saturated objects), which are excluded from the analysis.

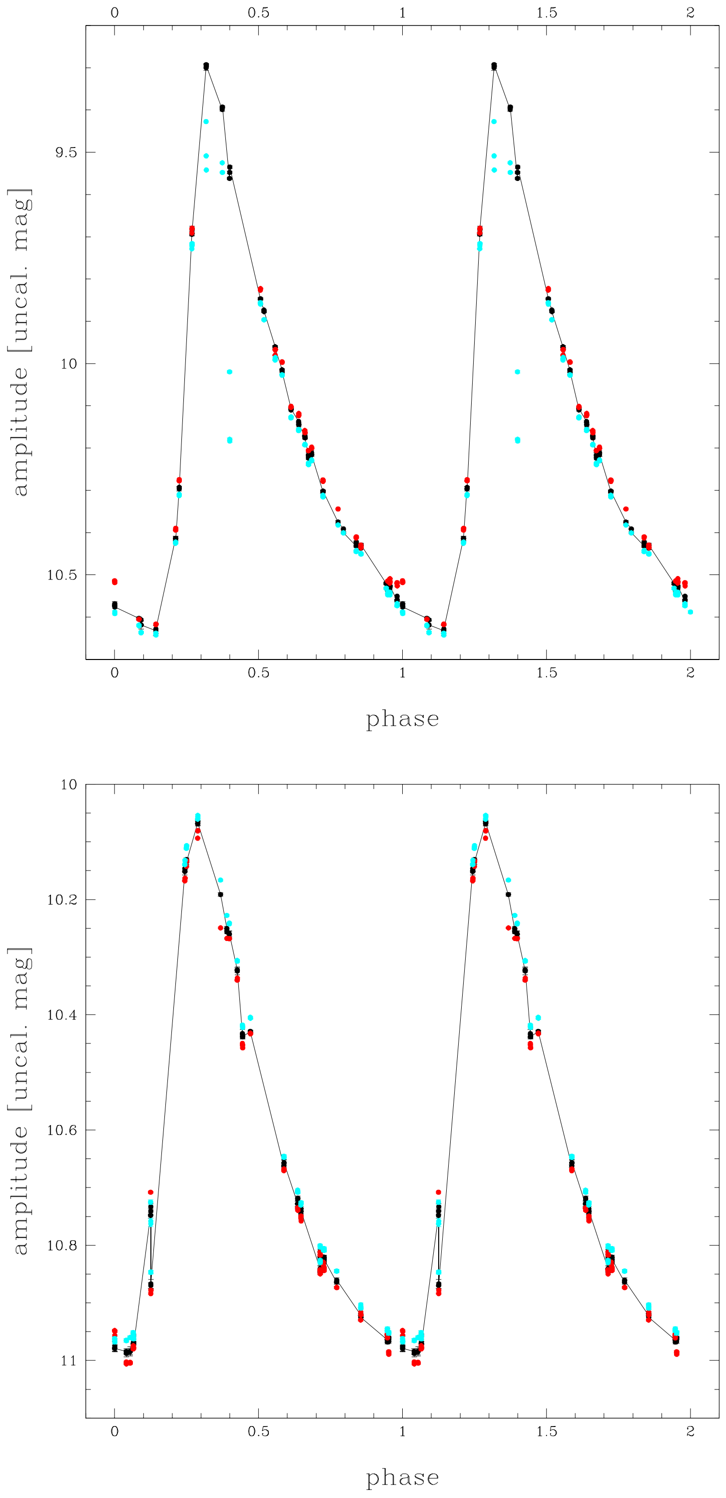
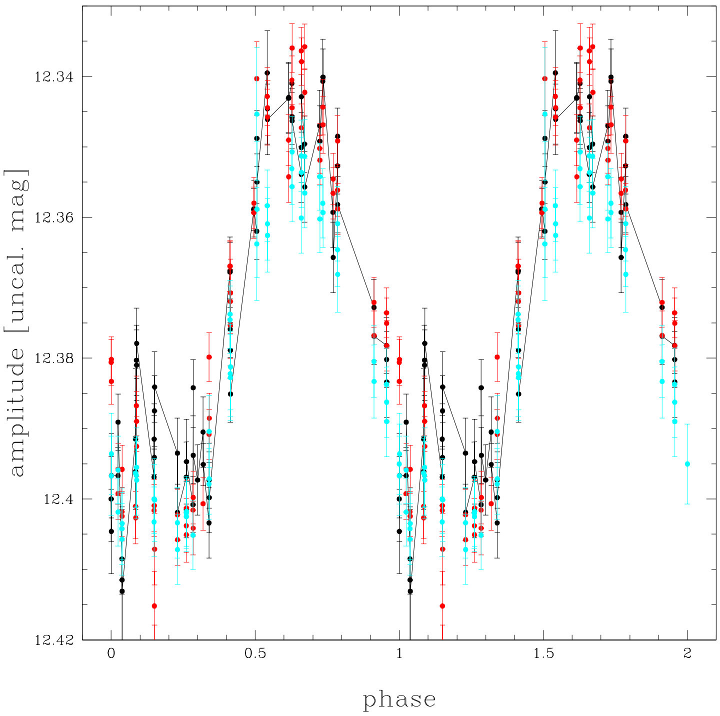


Figure 5.3: Comparison of the results obtained with VODIA (red dots) with those obtained using ALLFRAME (black) or using another software based on image subtraction (cyan). More details are given in the text. The y axis reports uncalibrated magnitudes (the offset to obtain calibrated magnitudes is of the order of ...). The error bars are not visible because they are smaller than the dimension of the dots.



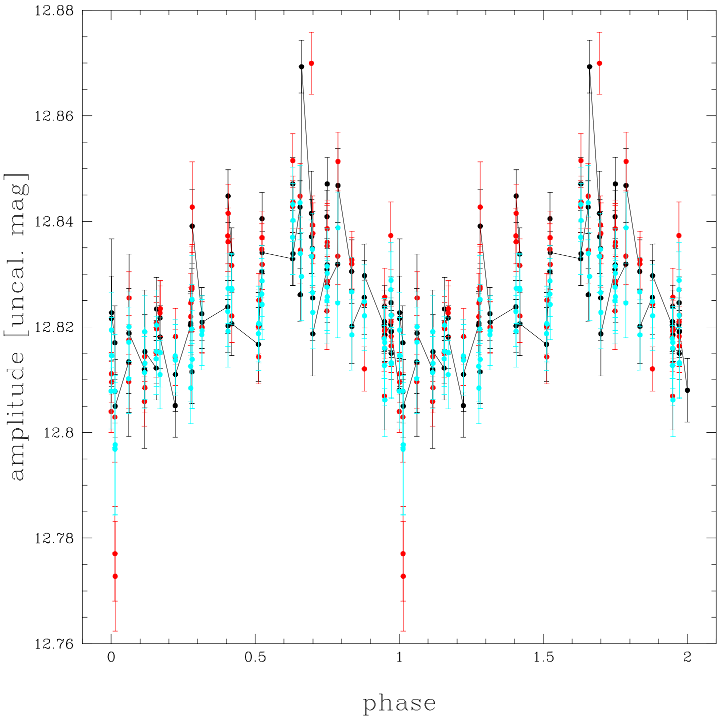


Figure 5.4: Same as Fig. 4 but for other two fainter variables located near the center of the cluster and having very small amplitudes.

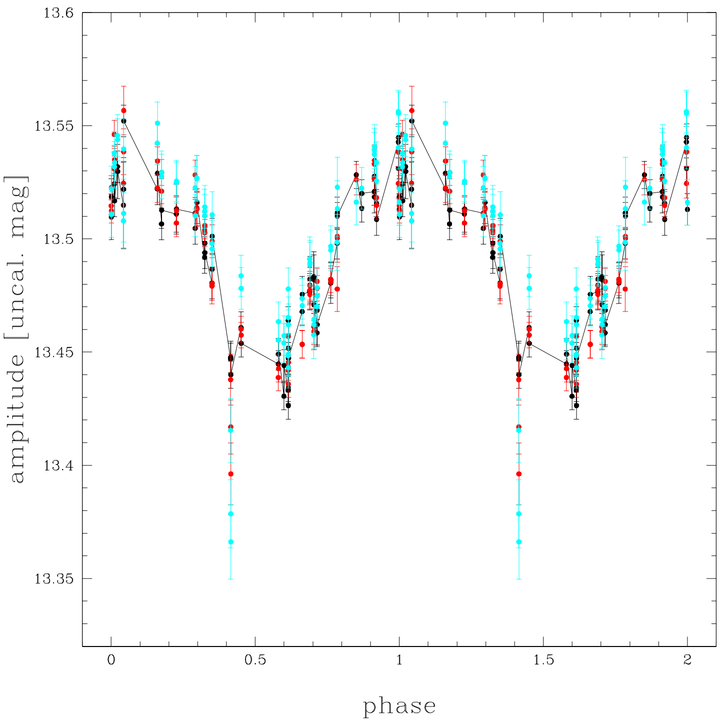


Figure 5.5: Same as Fig. 4 but for another very faint variable.

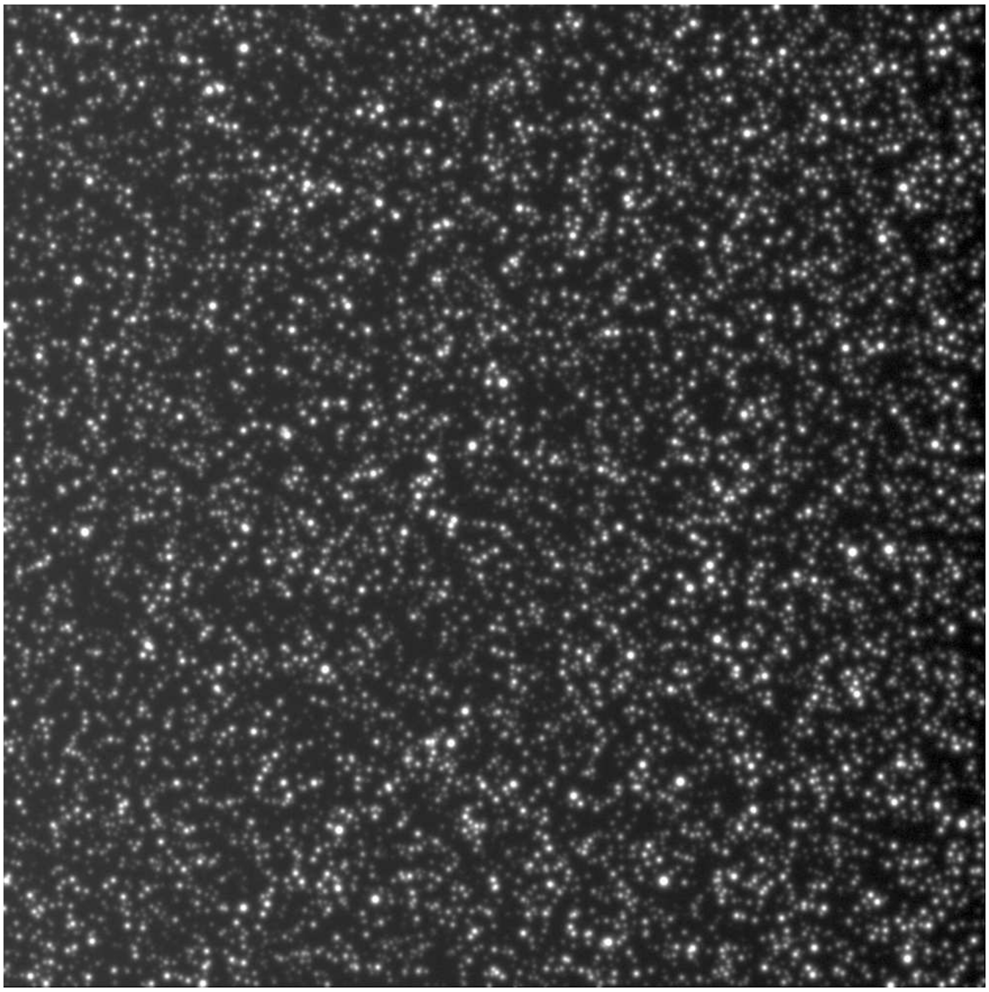
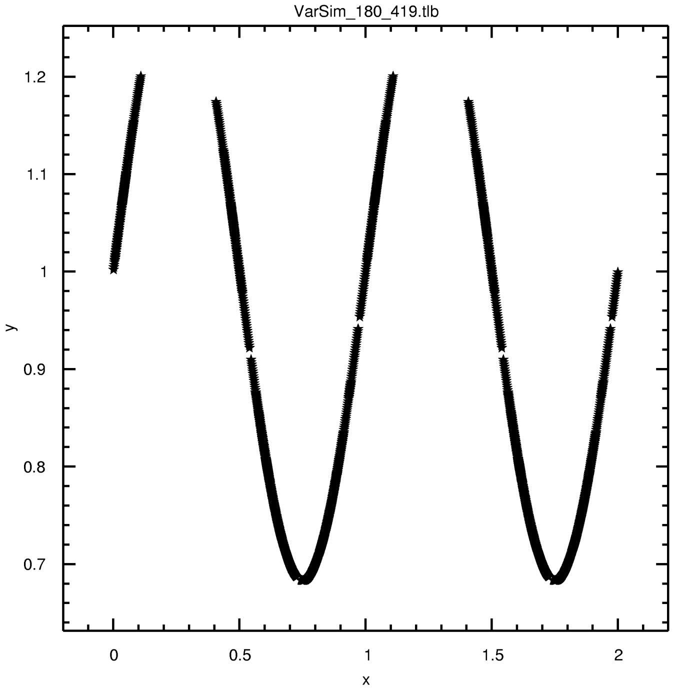


Figure 5.6: Simulated image.



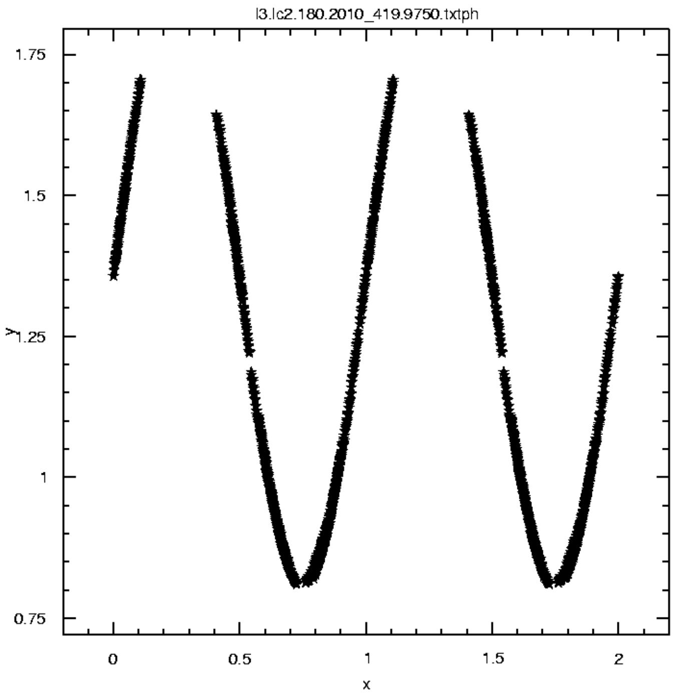


Figure 5.7: Comparison between the original simulated light curve without noise of a variable star (upper panel) and the results of VODIA (lower panel), after having extracted the light curve from the simulated images (which contain synthetic noise). Note that the relative amplitudes (y axis) are in good agreement.

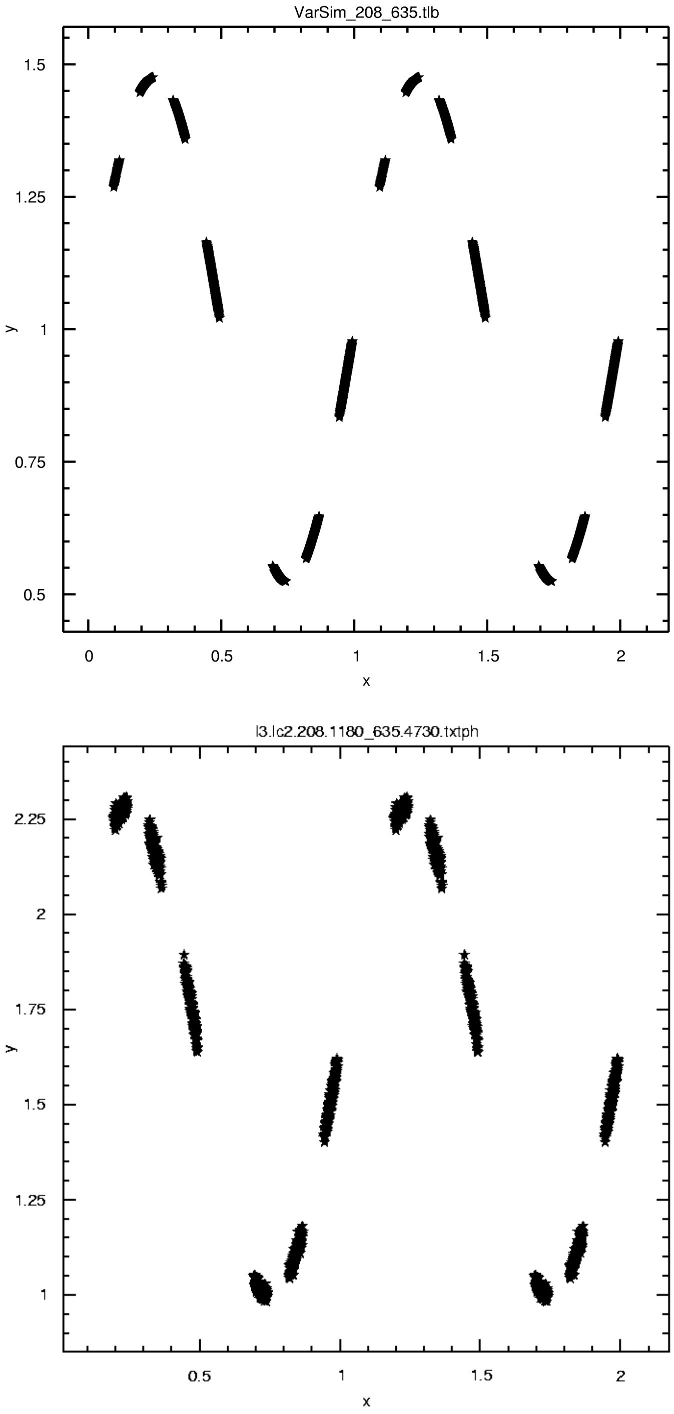


Figure 5.8: Same as Fig. 8 but for another variable star.

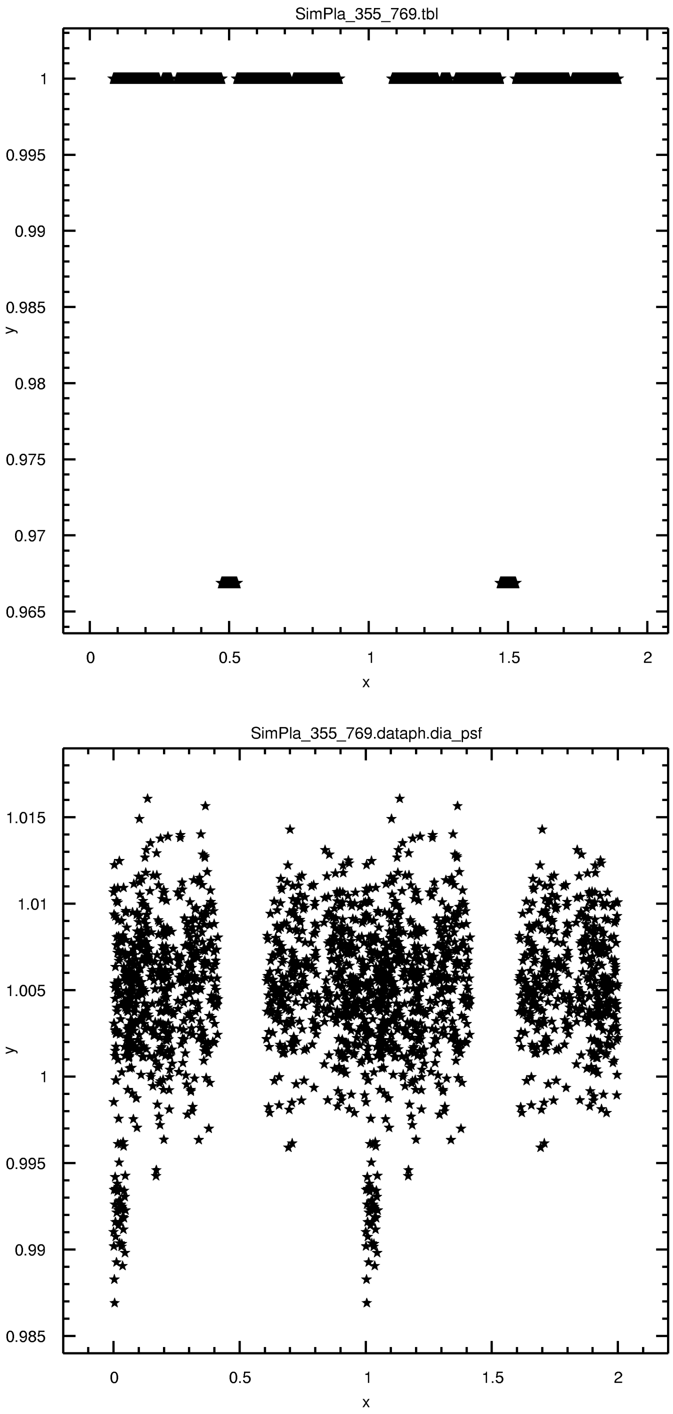


Figure 5.9: Same as Fig. 8 but for a planet transit. Here the difference in amplitude between the simulated light curve and the light curve exctracted from the simulated images is due to the very small amplitude of the occultation, which is comparable with the noise level. The phase shift (x axis) is simply due to an offset in phase.

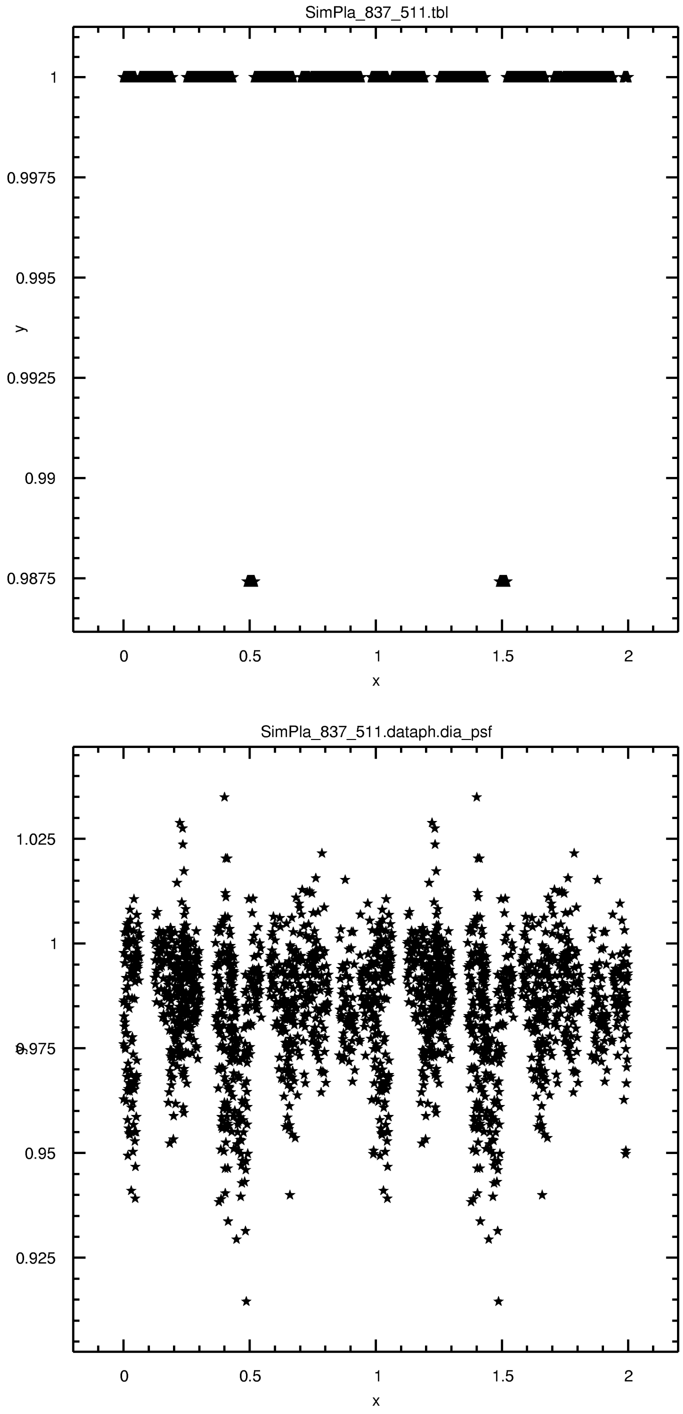


Figure 5.10: Same as Fig. 10 but for another planet occultation.

Chapter 6

Technical warnings and FAQs

6.1 Technical warnings

- VODIA was tested on 32 bit machines, no tests were done with 64 bit computers.
- VODIA requires images with no background subtraction.
- VODIA requires images with accurate astrometric solution and having same dimensions.
- VODIA depends on the memory of the machine in which it runs. In order to obtain the catalogue of the variable candidates, all the images are loaded in memory. Hence the user should take into account this aspect.
- In the present version VODIA requires that all the input images have the same exposure time.
- In the present version VODIA takes the following information from the headers, so beware that this information corresponds to the keywords listed below:

```
Number of pixels in the x axis     --->     NAXIS1
Number of pixels in the y axis     --->     NAXIS2
RA at ref pixel                    --->     CRVAL1
DEC at ref pixel                   --->     CRVAL2
Reference pixel in X               --->     CRPIX1
Reference pixel in Y               --->     CRPIX2
Translation matrix element         --->     CD1_1
Translation matrix element         --->     CD1_2
Translation matrix element         --->     CD2_1
Translation matrix element         --->     CD2_2
```

6.2 FAQ: solution to common problems running VODIA

This section contains a few comments and suggestions for the non-expert users, who want to use VODIA on their particular data set.

Which are the parameters that depend strongly on the type of data used and that have to be changed in order to obtain reliable results ?

- mstack.par

```
NX0
NY0
NY
BAD_VALUE
MIN_LEVEL
SAT_LEVEL
THRESHOLD
MEDIAN
```

- getpsf.par

```
NBOX_X
NBOX_Y
MIN_FLUX
MAX_THRESH
GAIN
SAT_LEVEL
MIN_LEVEL
```

- aga.par

```
NX_0
NY_0
NX
NY
GAIN
MIN_LEVEL
SAT_LEVEL
BAD_VALUE
```

- getvar.par

```
APRAD
FITRAD
ANRAD1
ANRAD2
SAT_LEVEL
MIN_LEVEL
BAD_VALUE
GAIN
EXPTIME
ERR_CODE
C_MIN
TMP_FILE
```

- phot.par

```
APRAD
FITRAD
ANRAD1
ANRAD2
BKG_MODE
SAT_LEVEL
MIN_LEVEL
BAD_VALUE
GAIN
EXPTIME
ERR_CODE
```

Running Aga the program stops with "segmentation fault" error message, what can I do?

- Check the list of images (it's ok? name of one image + name of the corresponding mask in each line?).
- Check that the names of the masks in the image list are different.
- Check the parameter file: NX0 and NY0 have good values according to your images?
- Check that the value of NX and NY parameters in the configuration file of Aga is less than $NX\_0 - 2 \times Ker\_HW$.
- Check that DOMAIN_MODE is not set to 2.

Running Aga the program stops without giving a kernel solution: each kernel is normalized to zero!! What can I do?

- Check the list of images (it's ok? name of one image + name of the corresponding mask in each line?).
- Check that the names of the masks in the image list are different.

How can I create a 16 bit mask from a 32 bit mask?

- We realized a program called float2short for this task. The usage is very simple:

```
> float2short <inputfile> <outputfile>

for example:

> float2short mask32bit.fits mask16bit.fits
```

If my data do not require masking, how can I use VODIA?

- Create a mask with all pixel values equal to one; (e.g. dividing the image for itself).
- Repeat item above for all your images giving a different name to each mask.

Aknowledgments

We want to thank P.R. Woźniak for usefull comments and suggestions.

References

Alard C., Lupton R.H., 1998, ApJ 503, 325

Alard C., 2000, A&A Suppl.Ser. 144, 363

Alcalá J.M., Pannella M., Puddu E. et al., 2004, A&A 428, 339

Dall'Ora M., Ripepi V., Caputo F. et al., 2003, AJ 126, 197

Woźniak P.R., 2000, Acta Astronomica 50, 421

Woźniak P.R.,Udalski A., Szymanski M. et al., 2002, Acta Astronomica 52, 129