

# Test procedures to validate the translation of the VISTA Active Optics MIST algorithm from Matlab to Python.

J. Kolb, 10/03/26

## 0. Test procedure

This document contains a list of procedures that were used to test the 17 Matlab routines required to compute the VISTA Active Optics commands from up to 4 WFS images. The procedures should be used to test the Python translation of the same routines, one by one.

It may not be necessary to run every single test, if the translation proves to be quickly good, but the individual tests may prove useful to debug differences between Matlab and Python.

Each chapter corresponds to a Matlab routine. In each chapter:

- There are several numbered tests
- Additional instructions are highlighted **in bold**
- Matlab command lines are preceded by the sign “>>”
- What follows is the Matlab reply to be compared with the Python one.
- Some test commands need to be translated to Python too, like the ones to display results: ‘plot’, ‘imagesc’, ‘figure’, ...
- When the reply is a larger map, it is shown together with the filename to be compared with the Python output. The filename is always the name of the Test, followed by ‘\_out’ (except for chapter 17). There may be ‘\_out2’, ... when there are more than one output to a test.
- When inputs are required for the test, the filename is provided. It is always the name of the Test, followed by ‘\_in’ (except for chapter 17).
- There are **highlights** for what is being tested, or the difference to previous tests

The results of the tests should be reported with crosses (“X”) and if needed comments in the following table:

Test ID	Executed	Passed	Comment
1.1			
1.2			
2.1			
2.2			
3.1			
3.2			

Test ID	Executed	Passed	Comment
3.3			
4.1			
4.2			
4.3			
5.1			
5.2			
5.3			
6.1			
6.2			
6.3			
6.4			
7.1			
7.2			
7.3			
7.4			
8.1			
8.2			
8.3			
8.4			
8.5			
9.1			
9.2			
9.3			
9.4			
9.5			
9.6			
10.1			
10.2			
10.3			
10.4			
10.5			
10.6			
10.7			
10.8			
10.9			
10.10			
11.1			
11.2			
11.3			
11.4			
11.5			
11.6			
12.1			
12.2			
12.3			

Test ID	Executed	Passed	Comment
12.4			
12.5			
12.6			
12.7			
13.1			
13.2			
13.3			
13.4			
14.1			
14.2			
15.1			
15.2			
16.1			
16.2			
16.3			
16.4			
17.1			
17.2			
17.3			
17.4			
17.5			
17.6			
17.7			
17.8			

## 1. Nansum

```
% y = nansum(x)
%
% Y = NANSUM(X) returns the sample sum of X, treating NaNs
as missing values.
% Custom sub-routines required:
% - none
```

### Test 1.1

```
>> A = [1 2 3 NaN 4 NaN 6]
A =
    1    2    3 NaN    4 NaN    6
>> sum(A)
ans =
    NaN
>> nansum(A)
ans =
    16
```

### Test 1.2

```
>> B = [[1 2 3 NaN]; [5 NaN 6 7]]
B =
    1    2    3 NaN
    5 NaN    6    7
>> sum(B)
ans =
    6 NaN    9 NaN
>> nansum(B)
ans =
    6    2    9    7
```

## 2. Nanmean

```
% m = nanmean(x,dim)
%
% M = NANMEAN(X) returns the sample mean of X, treating NaNs
as missing values.
% Custom sub-routines required:
% - none
```

### Test 2.1

```
>> A = [1 2 3 NaN 4 NaN 6]
A =
    1    2    3 NaN    4 NaN    6
>> mean(A)
ans =
    NaN
>> nanmean(A)
ans =
    3.2000
```

### Test 2.2

```
>> B = [[1 2 3 NaN];[5 NaN 6 7]]
B =
    1    2    3 NaN
    5 NaN    6    7
>> mean(B)
ans =
    3.0000    NaN    4.5000    NaN
>> nanmean(B)
ans =
    3.0000    2.0000    4.5000    7.0000
>> nanmean(B,2)
ans =
    2
    6
```

### 3. Proj

```
% Coeff = proj(A,B,C);  
%  
% This function computes the projection of the matrix 'A' on  
the matrix  
% 'B', in the sense of the least squares.  
% Custom sub-routines required:  
% 1) First level  
% - nanmean.m  
% - nansum.m
```

#### Test 3.1

```
>> B = [[1 2 3 NaN];[5 NaN 6 7]]  
B =  
    1    2    3 NaN  
    5 NaN    6    7  
>> C = [[10 9 8 7];[6 5 NaN 3]]  
C =  
    10    9    8    7  
     6    5 NaN    3  
>> proj(B,C)  
ans =  
   -0.7828
```

#### Test 3.2

```
>> proj(B,C)  
Error using proj (line 30)  
Input matrices must have the same size
```

#### Test 3.3

```
>> D = C - nanmean(C(:))  
D =  
    3.1429    2.1429    1.1429    0.1429  
   -0.8571   -1.8571     NaN   -3.8571  
>> E = nansum(nansum(D.*D))  
E =  
    34.8571  
>> proj(B,C,E)  
ans =  
   -0.7828
```

## 4. FindNM

```
% [nf,mf] = FindNM(mode);  
%  
% The goal of this function is to compute the index of radial  
% ('nf') and  
% azimuthal ('mf') orders of the Zernike polynomial of generic  
% order  
% 'mode'.  
% Custom sub-routines required:  
% - none
```

### Test 4.1

```
>> [nf,mf] = FindNM(12)
```

```
nf =
```

```
4
```

```
mf =
```

```
2
```

### Test 4.2

```
>> [nf,mf] = FindNM(103)
```

```
nf =
```

```
13
```

```
mf =
```

```
11
```

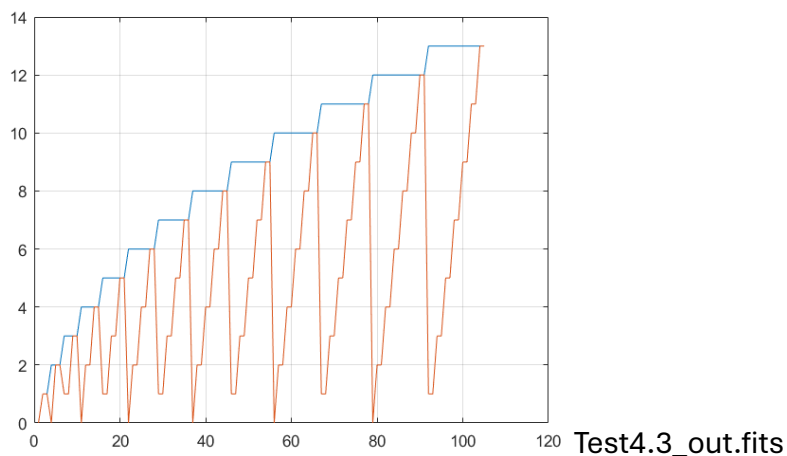
### Test 4.3

```
>> for cpt = 1:105
```

```
>> [nf(cpt),mf(cpt)] = FindNM(cpt);
```

```
>> end
```

```
>> figure ; plot(1:105,[nf;mf]) ; grid on
```



Test4.3\_out.fits

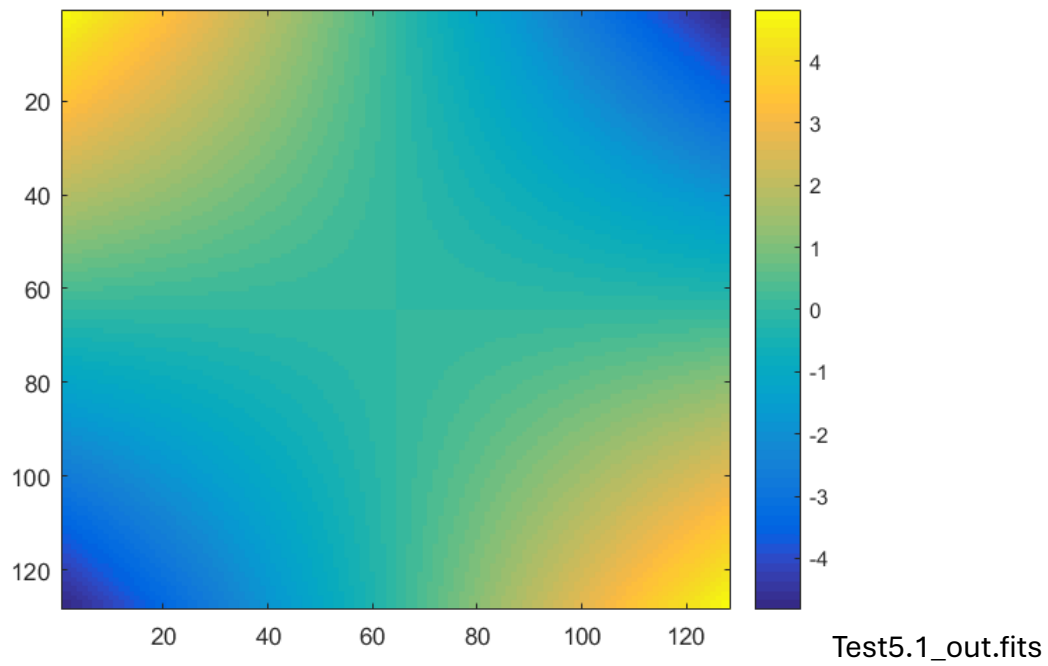
## 5. ZPolynomeQuick

```
% Zern = ZPolynomeQuick(mode,n,m,r,theta);  
%  
% The goal of this function is to compute the shape of a  
Zernike polynomial  
% Custom sub-routines required:  
% - none
```

### Test 5.1

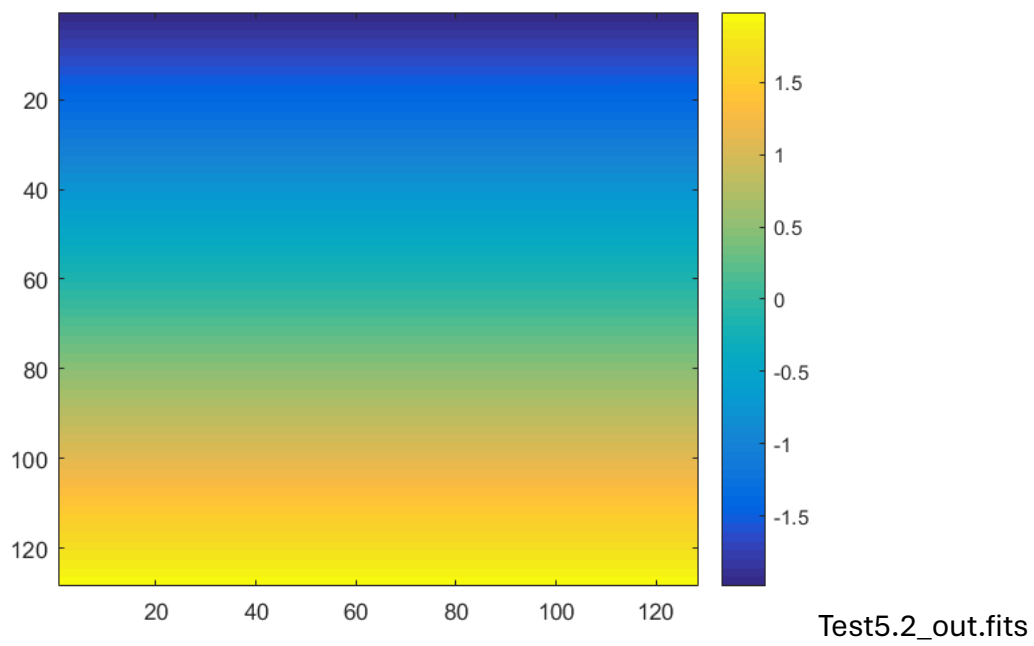
```
>> % Initialization  
>> nbpix = 128; cptZ = 5;  
>> Z = zeros(nbpix, nbpix);  
>> % Build a matrix of coordinates  
>> [X,Y] = meshgrid(-1+1/nbpix:(2-2/nbpix)/(nbpix-1):1-1/nbpix,-1+1/nbpix:(2-  
2/nbpix)/(nbpix-1):1-1/nbpix);  
>> % Convert into polar  
>> [THETA,RHO] = cart2pol(X,Y);  
>> % computes radial and orthogonal orders  
>> [nf,mf] = FindNM(cptZ);  
>> % calculation of the polynomial  
>> Zcpt = ZPolynomeQuick(cptZ,nf,mf,RHO,THETA);  
>> % population of the output matrix  
>> Z(:, :) = Zcpt;  
>> % display  
>> max(Z(:))  
ans =  
    4.8227  
>> std(Z(:))  
ans =  
    1.6329  
>> figure ; imagesc(Z) ; colorbar
```





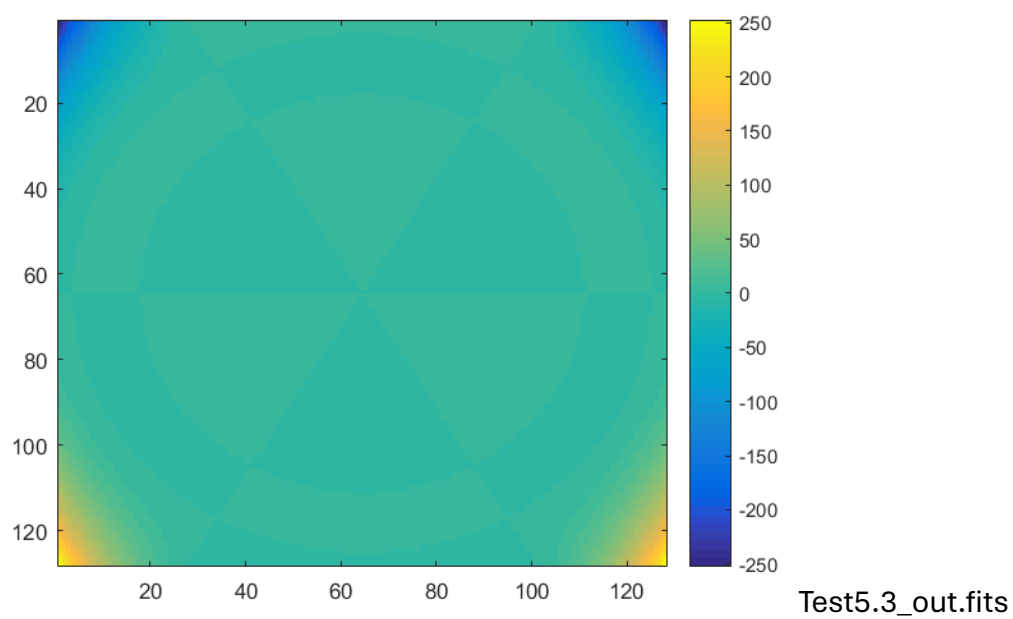
### Test 5.2

Same test but with `cptZ = 3;`



### Test 5.3

Same test but with `cptZ = 31;`

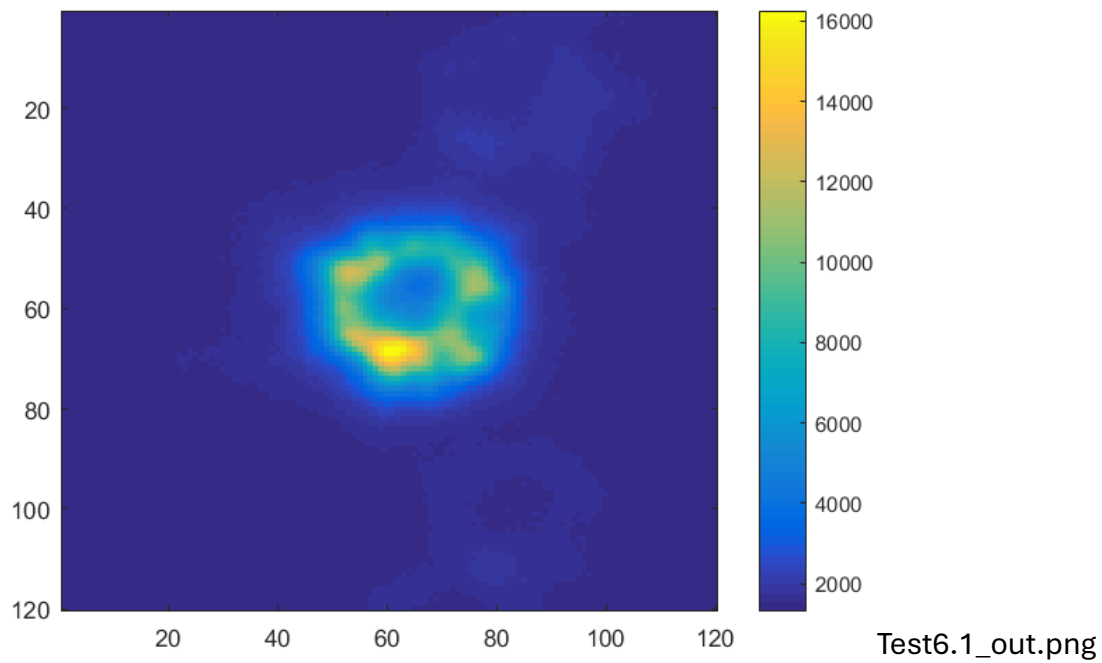


## 6. centroid\_PSIM

```
% [barxy, psf2] = centroid_PSIM(psf, Cen_meth, param);
% This function computes the centroid of an input image 'psf'
using the
% method 'Cen_meth' (and the parameter 'param') that can
either be:
%
% - 'Bar': simple Barycenter, in pixels, from the *corner* of the
image.
% In that case 'param' is the threshold value to be applied to
% the image before computing the CoG (without pedestal, i.e. the
% threshold value is removed from the image and then all negative
% values are set to zero). No threshold is applied if 'param' is
% not inputted.
% - 'CoG': Center of Gravity, in pixels from the *center* of the
image. Same threshold possibility as for 'Bar'
% - 'wCoG': weighted Center of Gravity. The input image is weighted
before computing the CoG. 'param' can then be a) the weighting map
and should be the same size as the input image, or b) the FWHM (in
% pixels) of the Gaussian spot to use as weighting function (i.e.
% a positive number), or c) the opposite of the power to which to
% take the input image (i.e. a negative number),
% If 'param' is not inputted, then the weighting map is the image
% itself, which is equivalent to squaring the image before
% computing the simple CoG.
% - 'Corr': Correlation. 'param' is then the FWHM (in pixels) of the
% Gaussian function to be used as correlation map.
% - 'MF': Matched filter. 'param' is then the filter map and should
% be the same size as the input image. NOT IMPLEMENTED YET
% - 'MGfit': Gaussian or Moffat fit. 'param' is then the fitting
method to be used, and should be set to:
% - 1 for regular Gaussian (requires 5 input parameters)
% - 2 for elliptical Gaussian (requires 7 input parameters)
% - 3 for regular Moffat (requires 6 input parameters)
% - 4 for elliptical Moffat (requires 8 input parameters)
%
% Custom sub-routines required:
% 1) First level
% - Corr_PSIM.m (only for 'Corr' input)
% - fminsearch (only for 'Corr' input)
% - MGfit_SPARTA (only for 'MGfit' input)
% 2) Other levels
% - MGfit_crit_SPARTA (only for 'MGfit' input)
% - crop_PSIM.m (only for 'MGfit' input)
```

### Test 6.1: basic barycenter

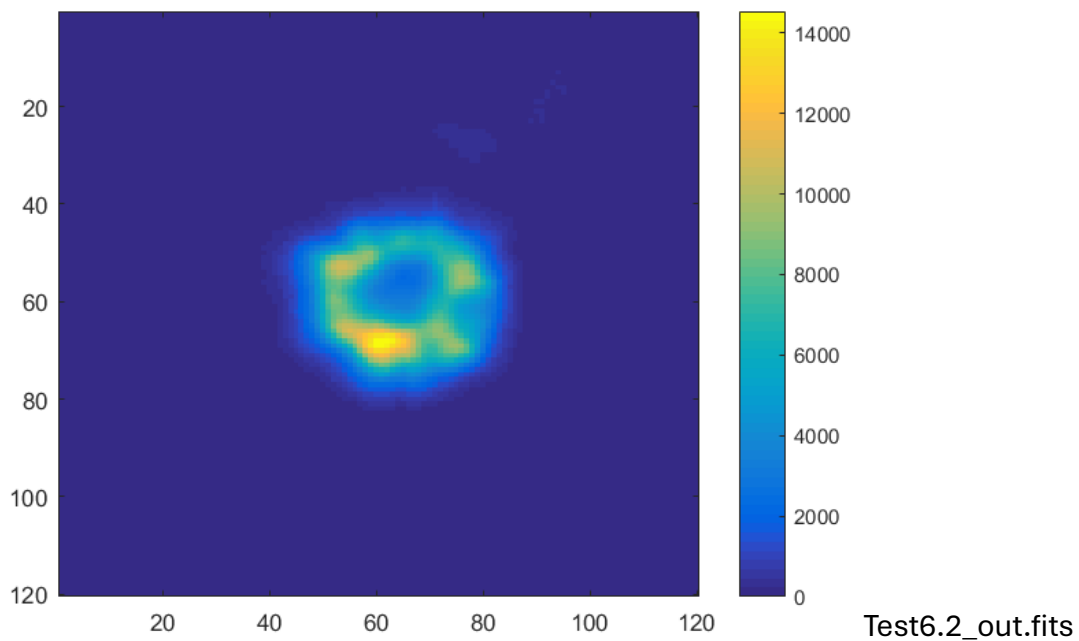
```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);
>> figure ; imagesc(Donut) ; colorbar
```



```
>> [barxy, psf2] = centroid_PSIM(Donut, 'Bar');
>> barxy
barxy =
    60.0983    62.1840
```

### Test 6.2: with input parameter

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'Bar', median(Donut(:))+3*std(Donut(1,1:end)));
>> barxy
barxy =
    59.9365    64.4700
>> figure ; imagesc(psf2) ; colorbar
```



### Test 6.3: Other centroiding methods

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'CoG'); barxy
barxy =
    -0.4017    1.6840
>> [barxy, psf2] = centroid_PSIM(Donut, 'wCoG'); barxy
barxy =
    0.0157    3.0008
>> [barxy, psf2] = centroid_PSIM(Donut, 'wCoG', 20); barxy
barxy =
    0.5702    0.4813
>> [barxy, psf2] = centroid_PSIM(Donut, 'Corr', 20); barxy
barxy =
    2.8176    2.8126
```

### Test 6.4: Other centroiding methods with parameters

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'Bar', -5); barxy
barxy =
    60.0983    62.1840
>> [barxy, psf2] = centroid_PSIM(Donut, 'Bar', 1000); barxy
barxy =
    59.6890    63.8998
>> [barxy, psf2] = centroid_PSIM(Donut, 'CoG', -10); barxy
barxy =
    -0.4017    1.6840
>> [barxy, psf2] = centroid_PSIM(Donut, 'CoG', 1000); barxy
barxy =
    -0.8110    3.3998
>> [barxy, psf2] = centroid_PSIM(Donut, 'wCoG', -3); barxy
barxy =
    1.1604    3.0018
>> [barxy, psf2] = centroid_PSIM(Donut, 'wCoG', Donut); barxy
barxy =
    0.0157    3.0008
>> [barxy, psf2] = centroid_PSIM(Donut, 'Corr', 30); barxy
barxy =
    0.5543    3.5179
```

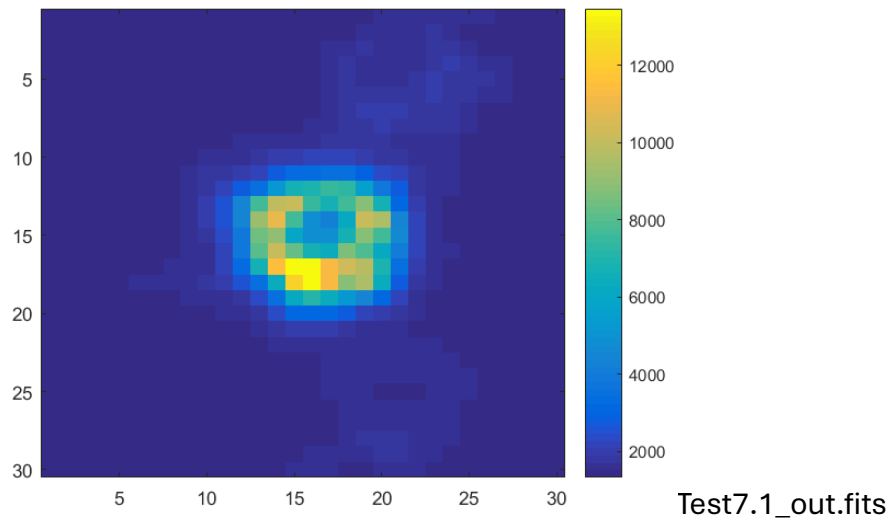
## 7. bin\_PSIM

```
% binned = bin_PSIM(pict, binfact);  
%  
% This function bins the input image 'pict' by the factor  
'binfact'.  
% Custom sub-routines required:  
% - none
```

### Test 7.1: binning with factor 4

Same image as for Test6.1:

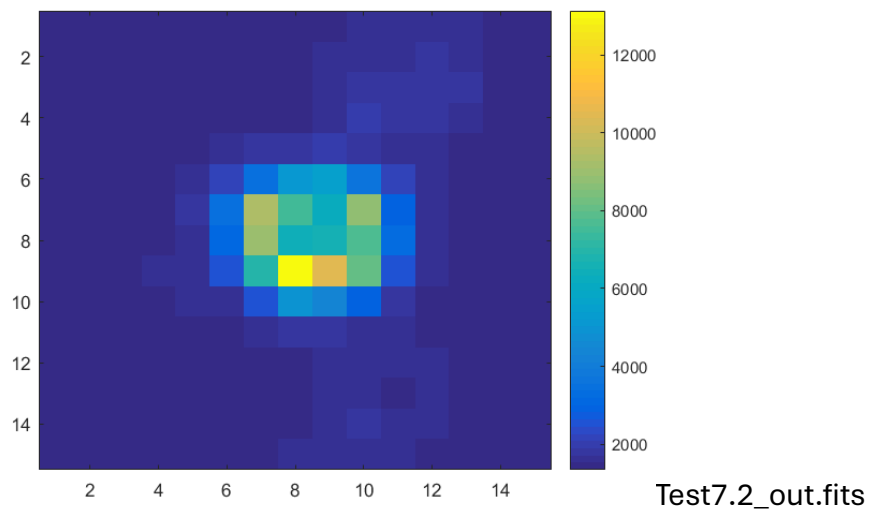
```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);  
>> Donut1 = bin_PSIM(Donut, 4);  
>> figure ; imagesc(Donut1) ; colorbar
```



```
>> max(Donut1(:))  
ans =  
1.3472e+04
```

### Test 7.2: binning with factor 8

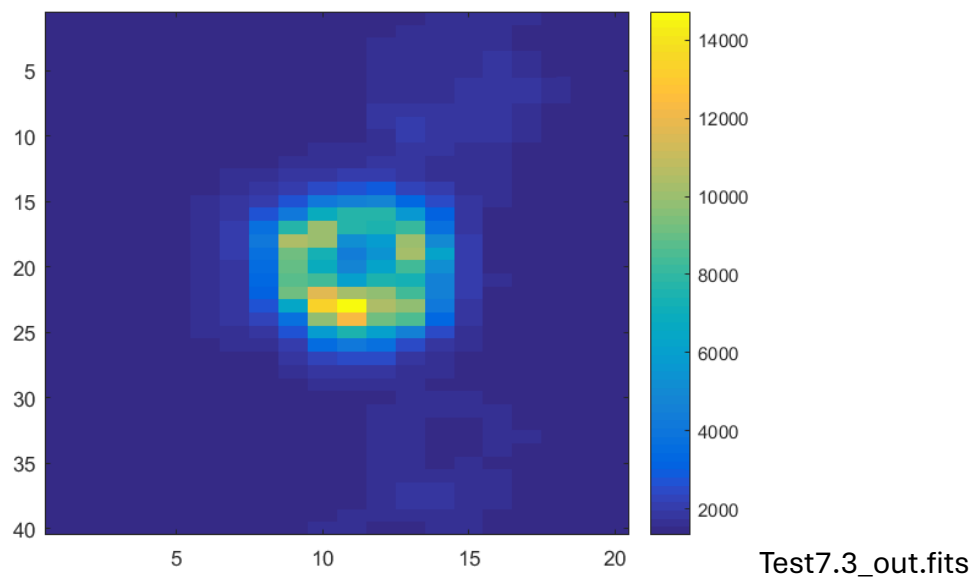
```
>> Donut2 = bin_PSIM(Donut, 8);  
>> figure ; imagesc(Donut2) ; colorbar
```



```
>> max(Donut2(:))
ans =
    1.3137e+04
```

### Test 7.3: non-square binning

```
>> Donut3 = bin_PSIM(Donut, [3 6]);
>> figure ; imagesc(Donut3) ; colorbar
```



```
>> max(Donut3(:))
ans =
    1.4728e+04
```

### Test 7.4: Error checking

```
>> Donut4 = bin_PSIM(Donut, [3 6 9]);
```

Error using bin\_PSIM (line 28)

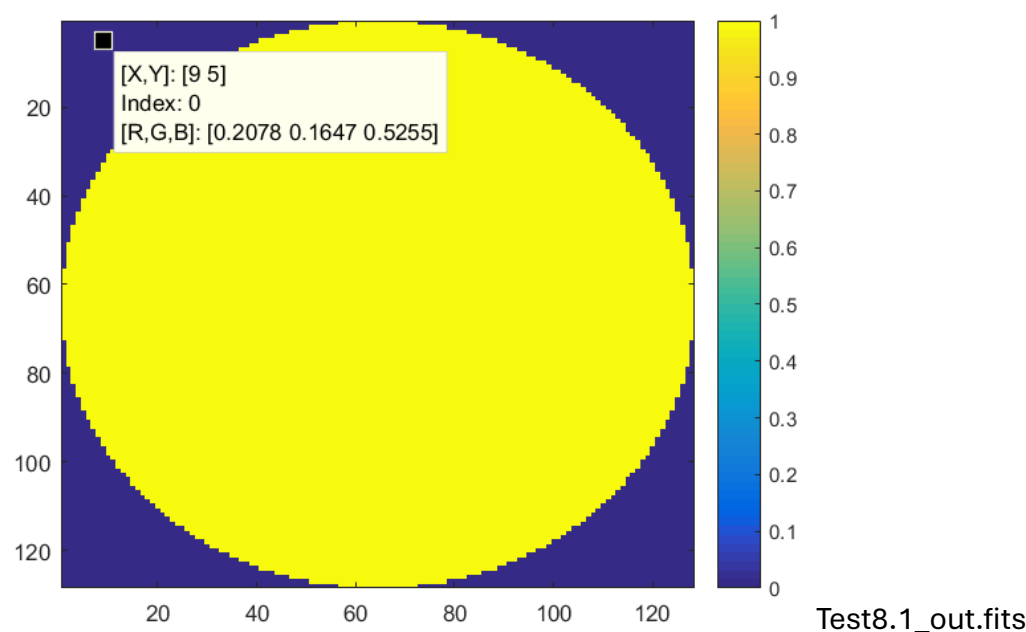
the input bin factor should contain 1 or 2 integer values

## 8. createZ\_PSIM

```
% [Z, in, out] = createZ_PSIM(nbpix, orders, innerD, outerD);  
%  
% The goal of this function is to generate Zernike  
% polynomials, computed  
% over a circular surface of radius 1, sampled over a square  
% grid of  
% pixels and cropped to the inputted inner and outer  
% diameters.  
% Custom sub-routines required:  
% 1) First level  
% - FindNM.m  
% - ZPolynomeQuick.m
```

### Test 8.1: Plain pupil with the number of pixels used in MIST (128)

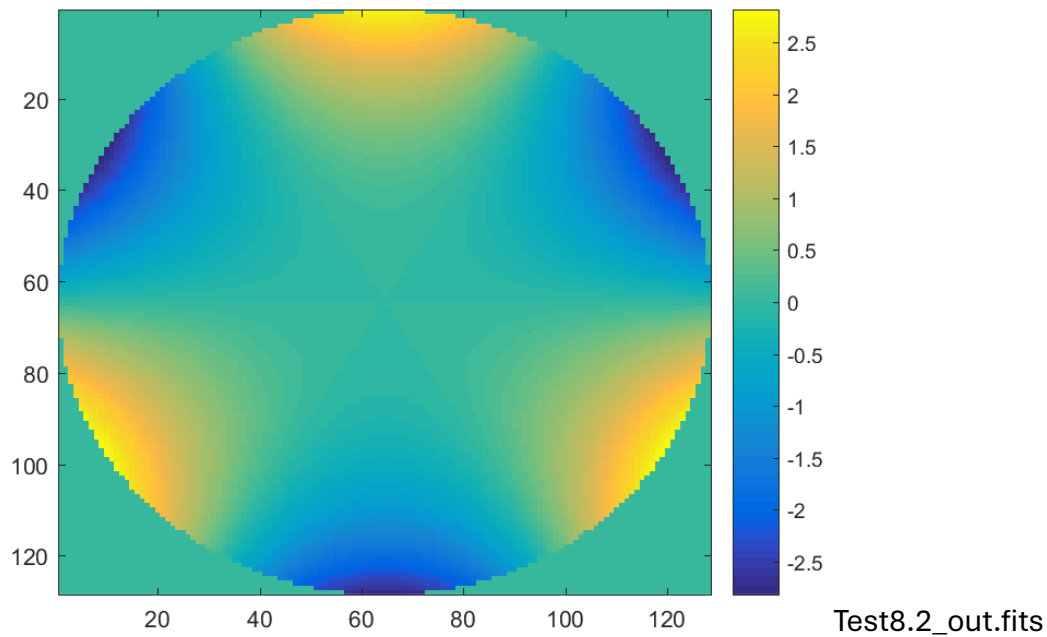
```
>> [Z, in, out] = createZ_PSIM(128, 1, 0, 1);  
>> std(Z(:))  
ans =  
    0.6980  
>> figure ; imagesc(Z) ; colorbar
```



### Test 8.2: Mode #9

```
>> [Z, in, out] = createZ_PSIM(128, 9, 0, 1);  
>> std(Z(:))  
ans =  
    0.8895  
>> figure ; imagesc(Z) ; colorbar
```





### Test 8.3: Small pupil (0.9)

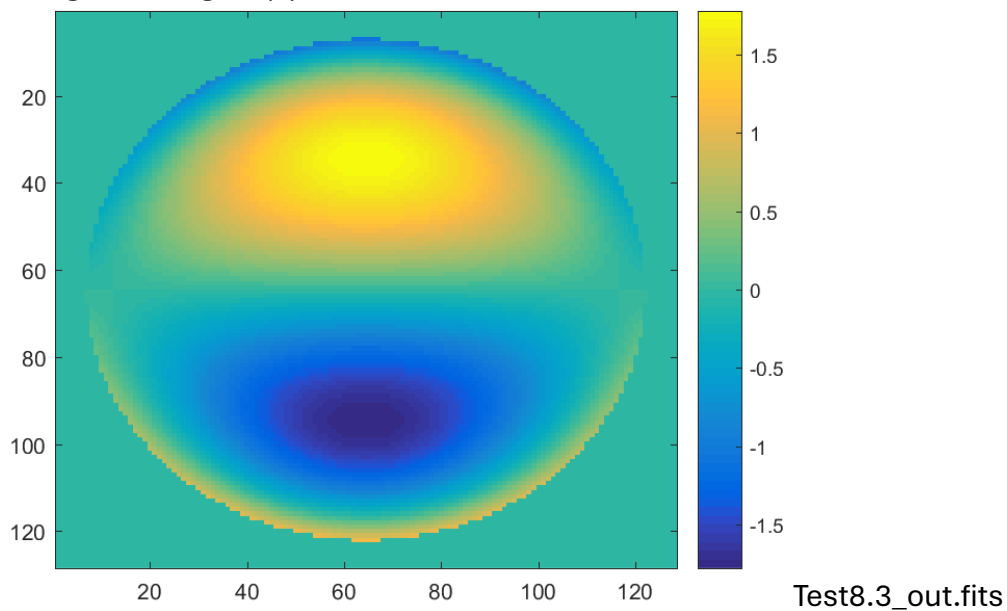
```
>> [Z, in, out] = createZ_PSIM(128, 7, 0, 0.9);
```

```
>> std(Z(:))
```

```
ans =
```

```
0.6980
```

```
>> figure ; imagesc(Z) ; colorbar
```



### Test 8.4: Central obscuration of VISTA (0.446)

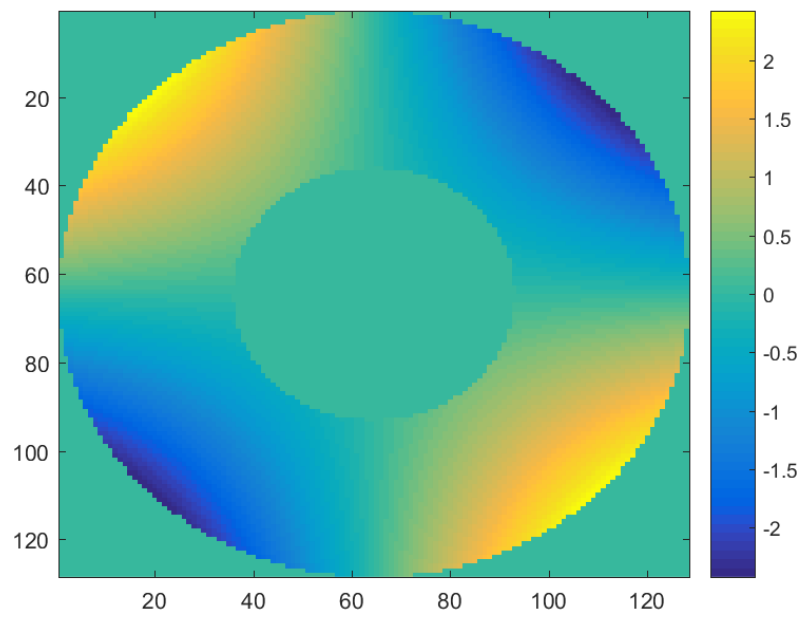
```
>> [Z, in, out] = createZ_PSIM(128, 5, 0.446, 1);
```

```
>> std(Z(:))
```

```
ans =
```

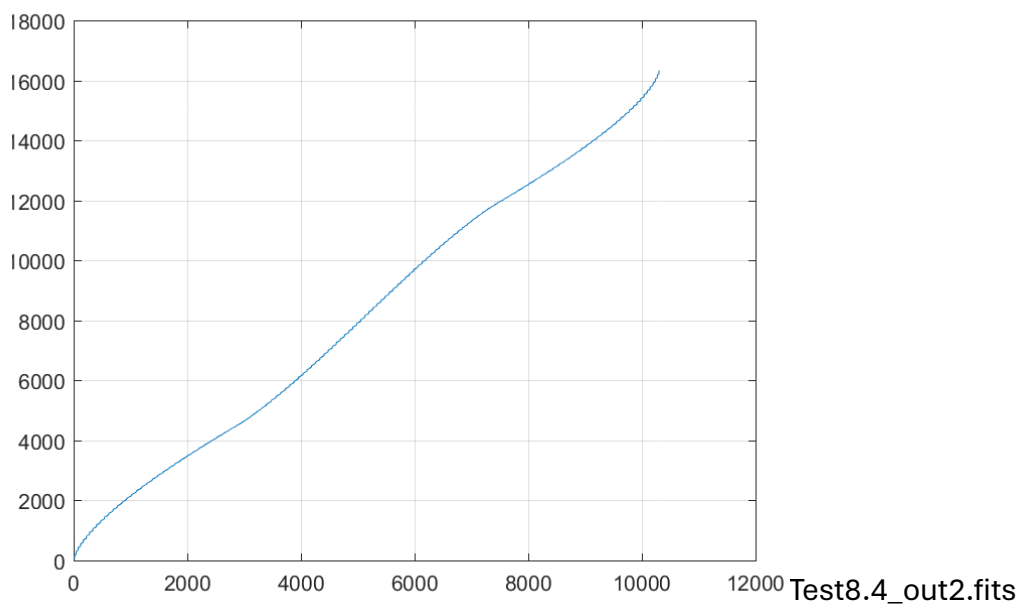
```
0.8851
```

```
>> figure ; imagesc(Z) ; colorbar
```



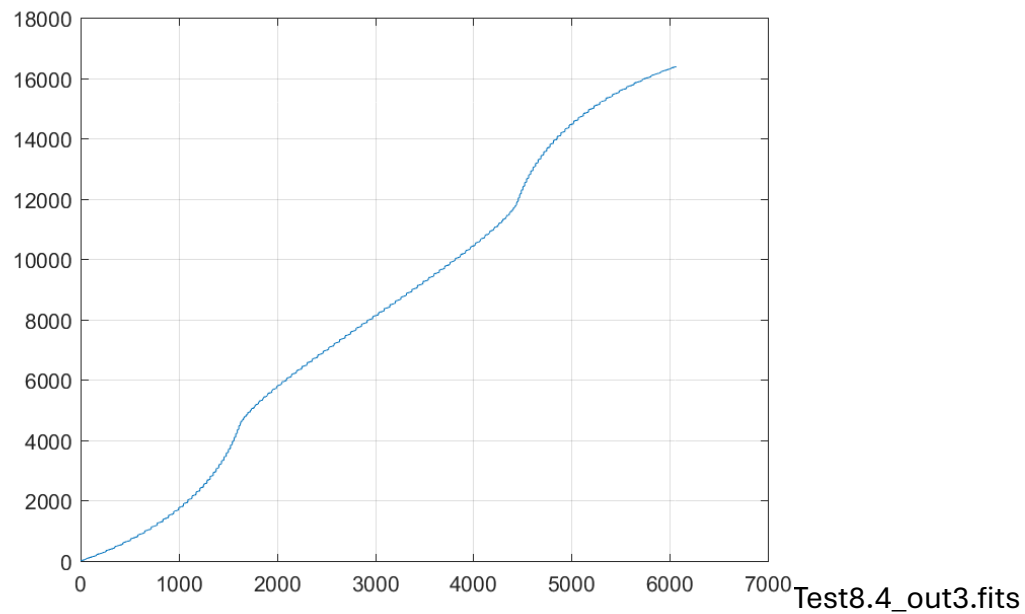
Test8.4\_out.fits

```
>> figure ; plot(in) ; grid on
```



Test8.4\_out2.fits

```
>> figure ; plot(out) ; grid on
```



### Test 8.5: Cube of Zernike modes

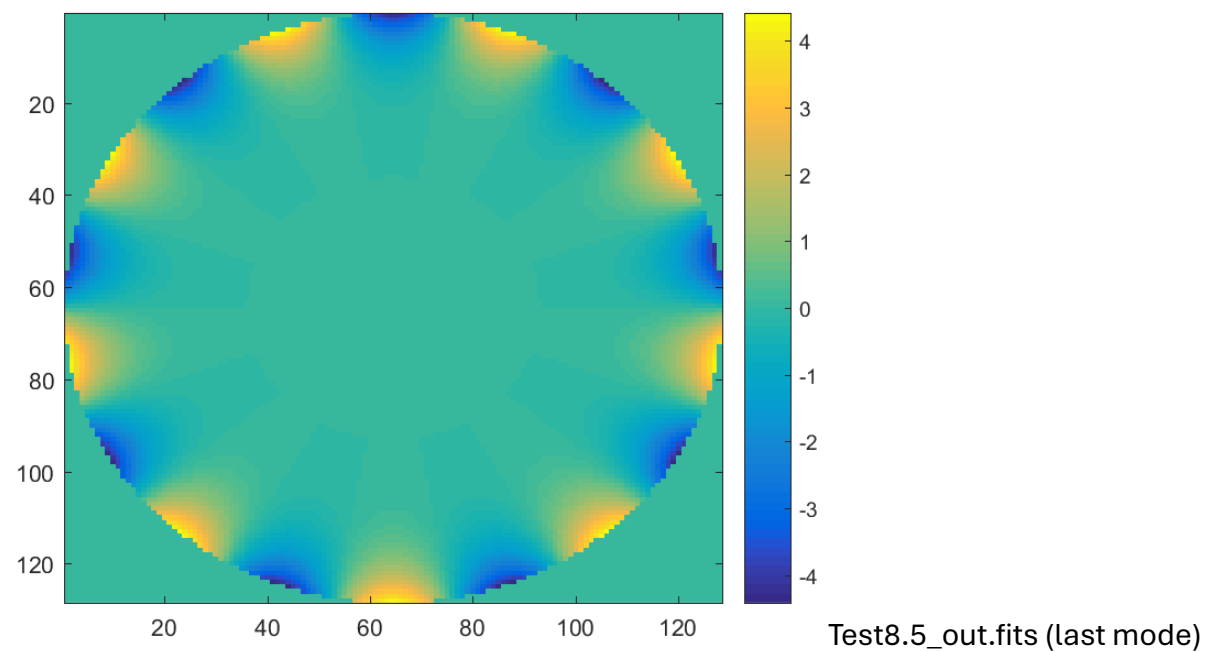
```
>> [Z, in, out] = createZ_PSIM(128, 1:55, 0.446, 1);
```

```
>> std(Z(:))
```

```
ans =
```

```
0.8436
```

```
>> figure ; imagesc(Z(:, :, 55)) ; colorbar
```



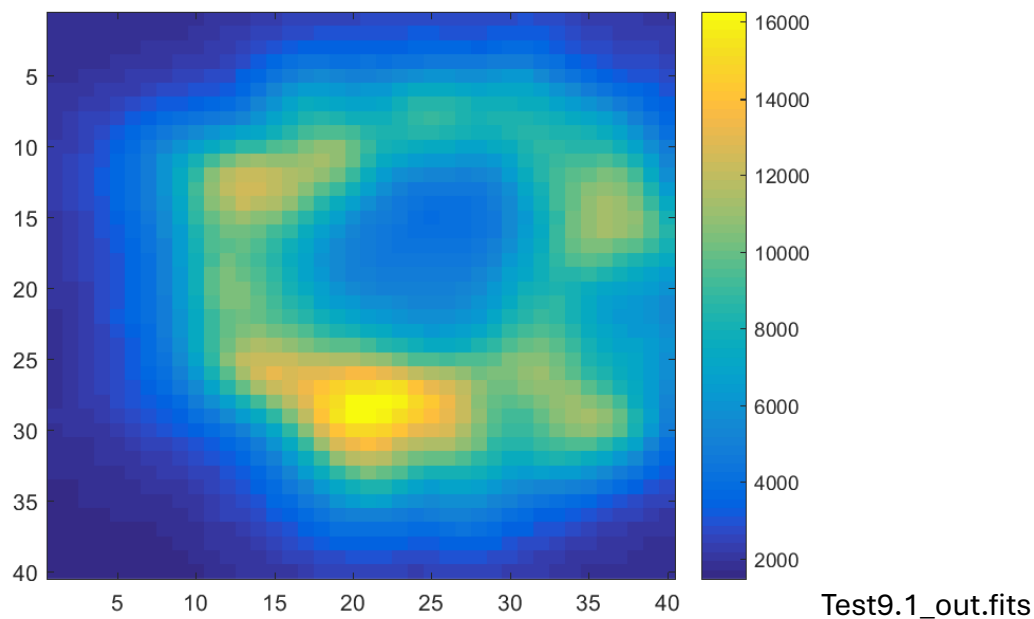
## 9. crop\_MIST

```
% [psf2, startxy, endxy] = crop_PSIM(psf, sizexy, barxy);  
%  
% This function crops a window in the input image 'psf'.  
% Custom sub-routines required:  
% 1) First level  
% - centroid_PSIM.m
```

### Test 9.1: square crop from center

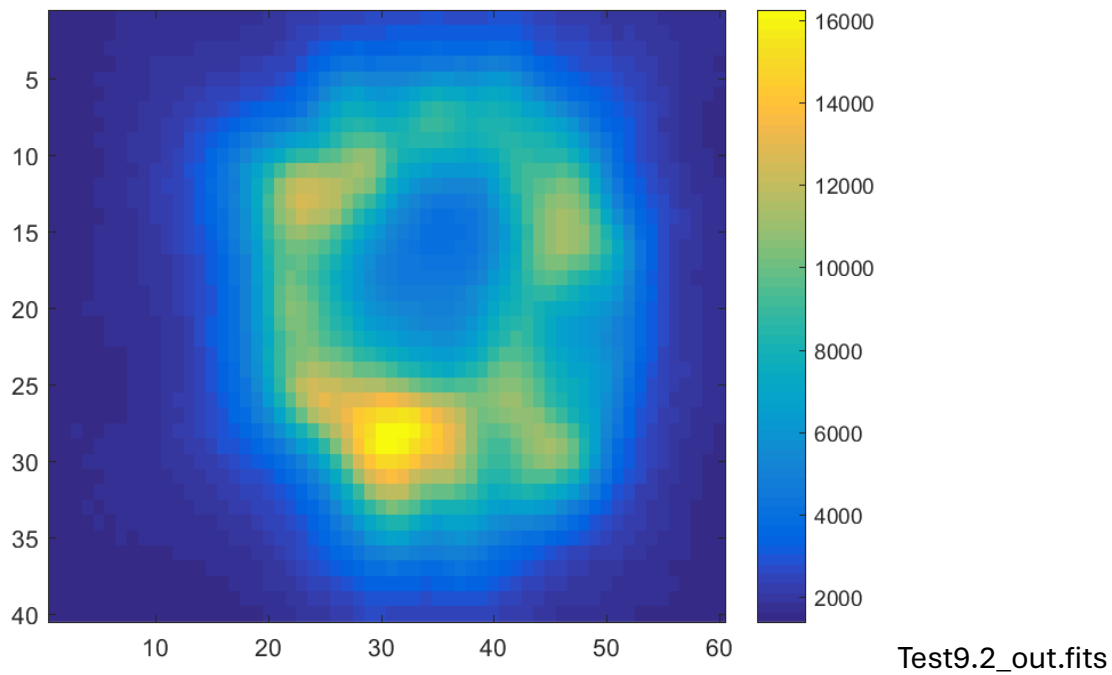
Same image as for Test6.1:

```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);  
>> [Donut2, startxy, endxy] = crop_MIST(Donut, 40);  
>> std(Donut2(:))  
ans =  
    3.3340e+03  
>> figure ; imagesc(Donut2) ; colorbar
```



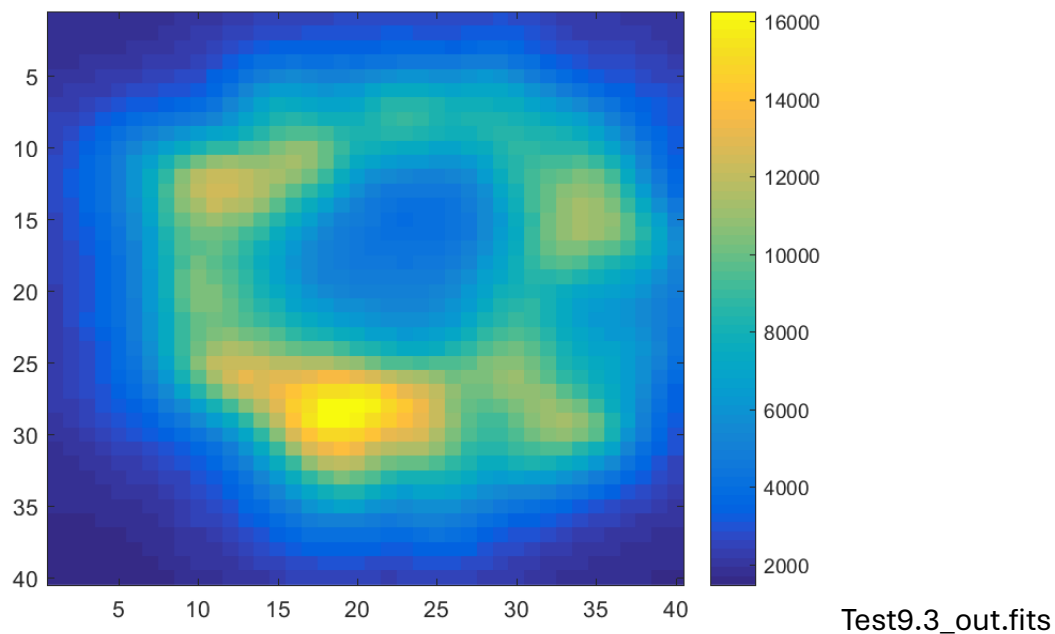
### Test 9.2: rectangular crop from center

```
>> [Donut2, startxy, endxy] = crop_MIST(Donut, [40,60]);  
>> std(Donut2(:))  
ans =  
    3.3004e+03  
>> figure ; imagesc(Donut2) ; colorbar
```



### Test 9.3: square crop from barycenter

```
>> [Donut2, startxy, endxy] = crop_MIST(Donut, 40, -1);
>> std(Donut2(:))
ans =
    3.2713e+03
>> figure ; imagesc(Donut2) ; colorbar
```



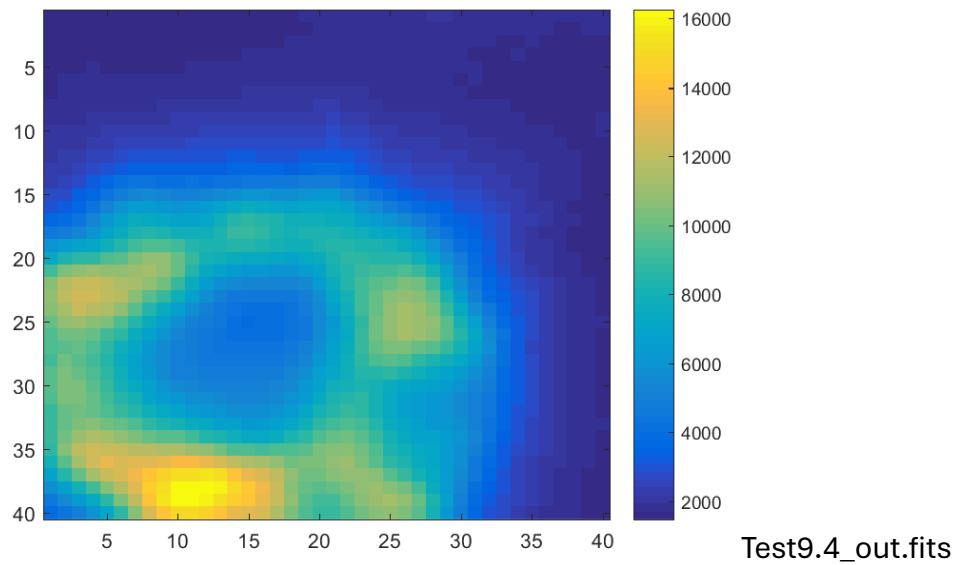
### Test 9.4: offset square crop

```
>> [Donut2, startxy, endxy] = crop_MIST(Donut, 40, [50, 70]);
>> std(Donut2(:))
```

```
ans =
```

```
3.5507e+03
```

```
>> figure ; imagesc(Donut2) ; colorbar
```



**Test 9.5: outputs, from Test 9.4:**

```
>> startxy
```

```
startxy =
```

```
31 51
```

```
>> endxy
```

```
endxy =
```

```
70 90
```

**Test 9.6: Error checking**

```
>> [Donut2, startxy, endxy] = crop_MIST(Donut, 60.5);
```

Error using crop\_MIST (line 50)

output image size must be integer

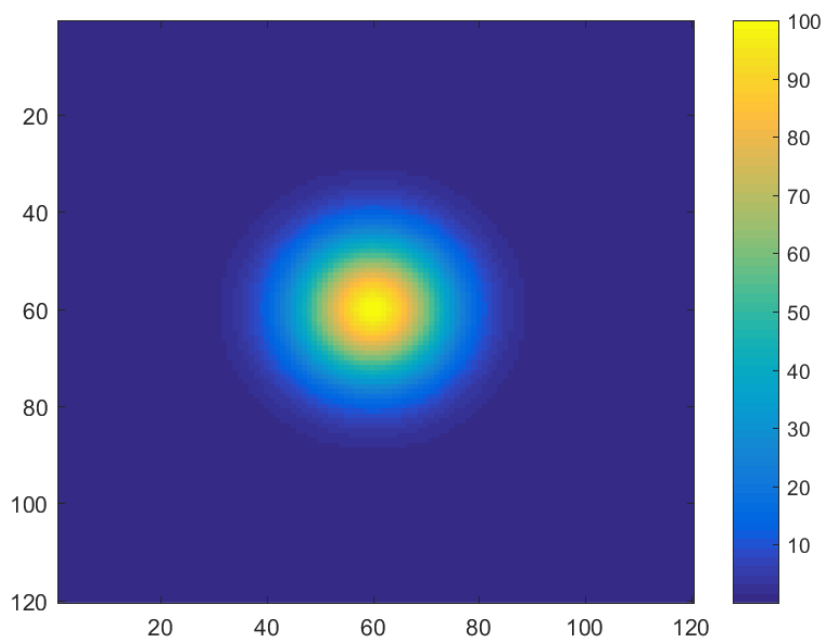
## 10. MGfit\_crit\_SPARTA

```
% [Crit, PSF2, FWHM] = MGfit_crit_SPARTA(PSF, p, fit_meth,  
psf_dl, forced_fit, Cmin, X, Y)  
%  
% This function generates a Gaussian or Moffat PSF from the  
input  
% parameters, and computes a criterion equivalent to the rms  
of the  
% difference between an input PSF and this model one. NaNs are  
ignored. The  
% FWHM of the model PSF is also computed.  
%  
% - 'fit_meth' is the fitting method to be used, and should be set  
to:  
%   - 1 for regular Gaussian (requires 5 input parameters)  
%   - 2 for elliptical Gaussian (requires 7 input parameters)  
%   - 3 for regular Moffat (requires 6 input parameters)  
%   - 4 for elliptical Moffat (requires 8 input parameters)  
  
% - 'p' a vector that contains the input parameters  
%   - p(1) is the offset  
%   - p(2) is the amplitude  
%   - p(3:4) are the coordinates of the center of the PSF, in pixels  
%   - for a circular Gaussian fit, p(5) is the sigma of the  
Gaussian, in  
%     pixels  
%   - for an elliptical Gaussian fit, p(5) is the sigma of the  
Gaussian  
%     along the minor axis in pixels, p(6) the additional sigma (to  
p(5))  
%     to get the major axis in pixels, and p(7) is the orientation  
angle in  
%     radians  
%   - for a circular Moffat fit, p(5) is the sigma of the function  
in  
%     pixels and p(6) the beta coefficient (in the exponent)  
%   - for an elliptical Moffat fit, p(5) is the sigma of the  
function  
%     along the minor axis in pixels, p(6) the additional sigma (to  
p(5))  
%     to get the major axis in pixels, p(7) is the orientation angle  
in  
%     radians and p(8) the beta coefficient (in the exponent).  
  
% Custom sub-routines required:  
% 1) First level  
% - crop_PSIM.m  
% - conv2.m (Matlab function)
```

### Test 10.1: Generation of a circular Gaussian (5 input parameters)

Same image as for Test6.1 (in fact inputted just to get the image size, and compare with the generated PSF to get the 'Crit' output)

```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);  
>> [Crit, Donut1, FWHM] = MGfit_crit_SPARTA(Donut, [0 100 60 60 10], 1, 0);  
>> Crit  
Crit =  
    2.6498e+03  
>> FWHM  
FWHM =  
    23.5482  
>> figure ; imagesc(Donut1) ; colorbar
```



### Test 10.2: Check 'Crit' when input image is the same as output

```
>> [Crit, Donut2, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 60 60 10], 1, 0);  
>> Crit  
Crit =  
    0
```

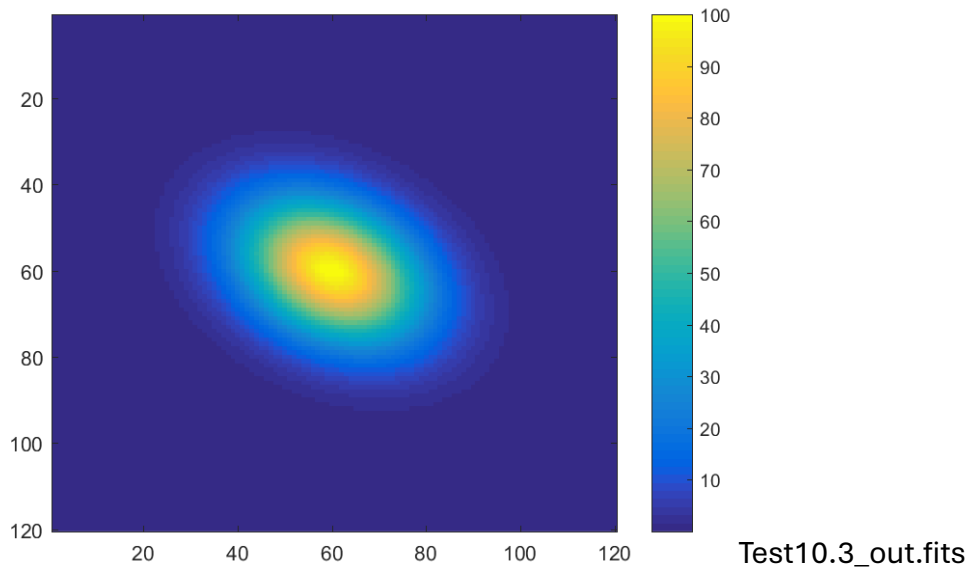
### Test 10.3: Generation of an elongated Gaussian (7 input parameters) and comparison with the circular one

```
>> [Crit, Donut3, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 60 60 10 4 45], 2, 0);  
>> Crit  
Crit =  
    4.6336  
>> FWHM  
FWHM =
```



23.5482 32.9675

```
>> figure ; imagesc(Donut3) ; colorbar
```



**Test 10.4: Generation of a circular Moffat (6 input parameters) and comparison with the circular one**

```
>> [Crit, Donut4, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 60 60 10 1.5], 3, 0);
```

```
>> Crit
```

Crit =

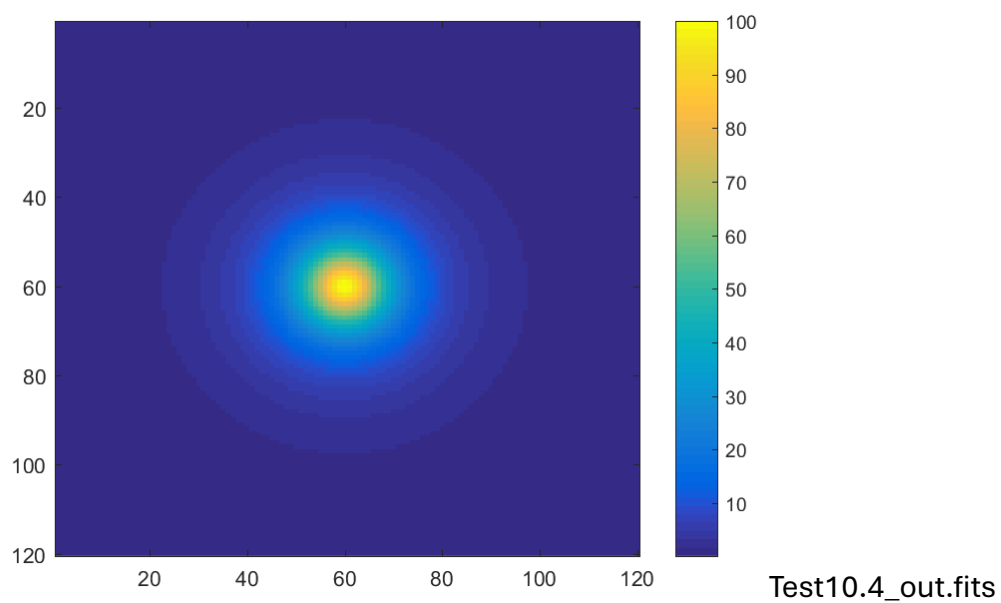
5.1689

```
>> FWHM
```

FWHM =

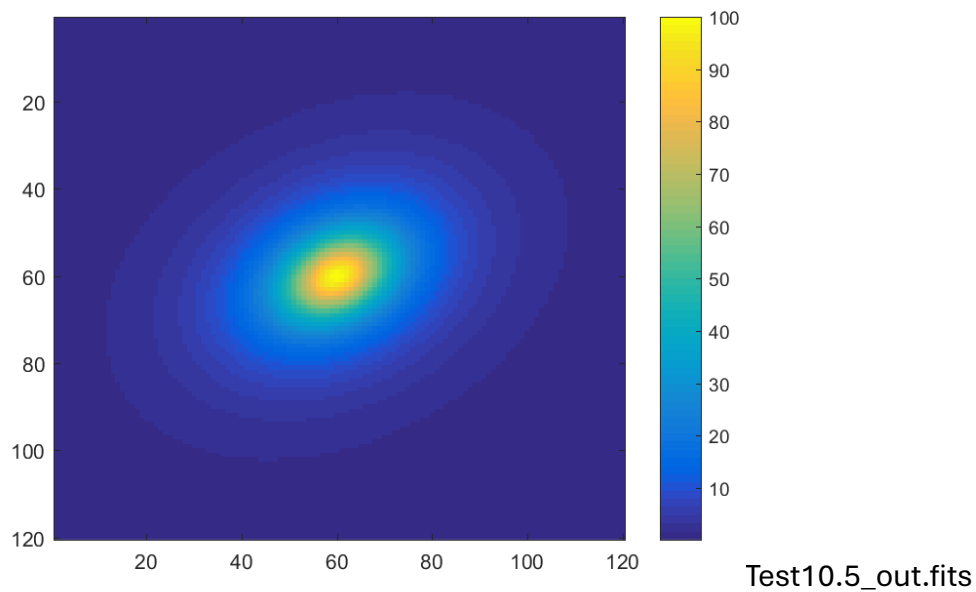
15.3284

```
>> figure ; imagesc(Donut4) ; colorbar
```



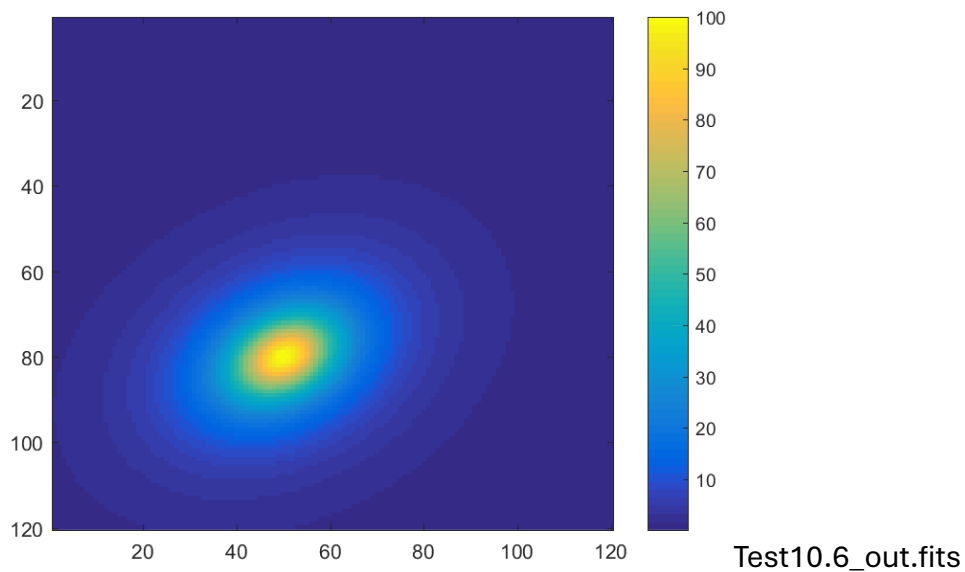
**Test 10.5: Generation of an elongated Moffat (8 input parameters) and comparison with the Gaussian circular one**

```
>> [Crit, Donut5, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 60 60 10 4 -45 1.5], 4, 0);  
>> Crit  
Crit =  
    4.0992  
>> FWHM  
FWHM =  
    15.3284    21.4598  
>> figure ; imagesc(Donut5) ; colorbar
```



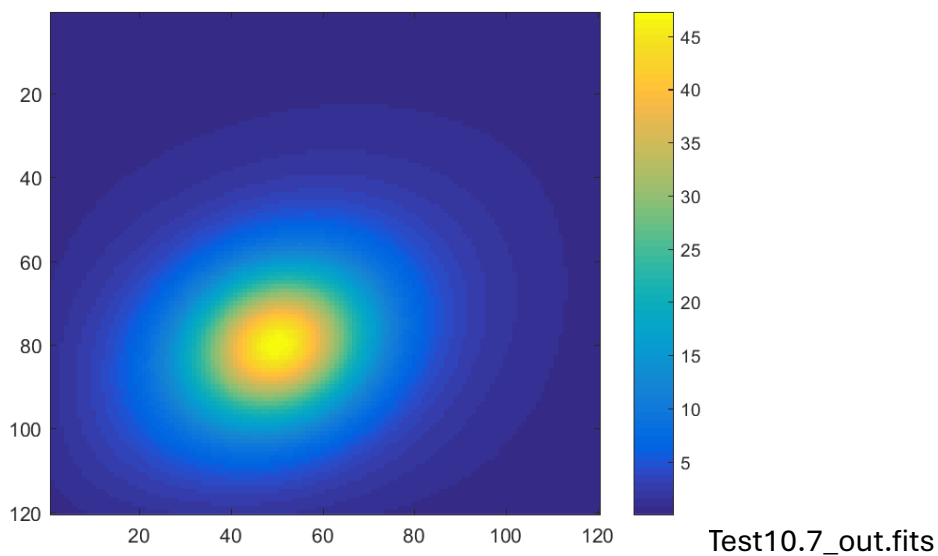
**Test 10.6: Generation of an offset elongated Moffat and comparison with the Gaussian circular one**

```
>> [Crit, Donut6, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 50 80 10 4 -45 1.5], 4, 0);  
>> Crit  
Crit =  
    15.6155  
>> FWHM  
FWHM =  
    15.3284    21.4598  
>> figure ; imagesc(Donut6) ; colorbar
```



### Test 10.7: Convolution with an extended source

```
>> [Crit, Donut7, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 50 80 10 4 -45 1.5], 4, 20);
>> Crit
Crit =
    13.6551
>> FWHM
FWHM =
    15.3284    21.4598
>> figure ; imagesc(Donut7) ; colorbar
```



### Test 10.8: Error checking when convolving with an extended source

```
>> [Crit, Donut8, FWHM] = MGfit_crit_SPARTA(Donut1, [0 100 50 80 10 4 -45 1.5], 4, 10);
```

Error using MGfit\_crit\_SPARTA (line 126)

"psf\_dl" input must be a scalar or an array of the same size as the input psf

### Test 10.9: Over-sampling (Cmin = 4) and comparison with Cmin = 1

```
>> [Crit, Donut9, FWHM] = MGfit_crit_SPARTA(Donut6, [0 100 50 80 10 4 -45 1.5], 4, 0, 0, 4);
```

```
>> Crit
```

```
Crit =
```

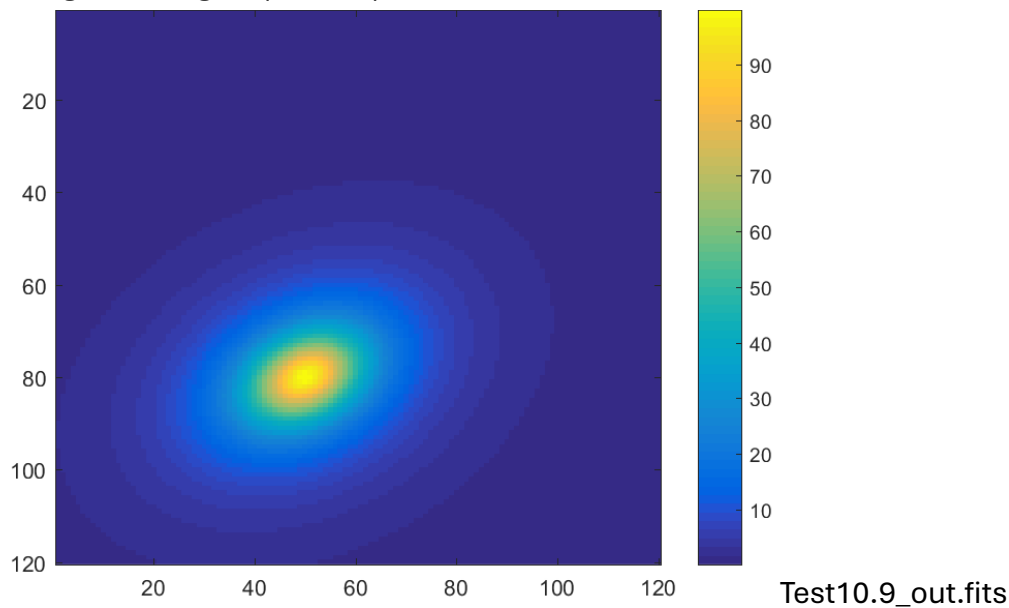
```
0.0102
```

```
>> FWHM
```

```
FWHM =
```

```
15.3284 21.4598
```

```
>> figure ; imagesc(Donut9) ; colorbar
```



### Test 10.10: Inputting the sampling grid for speed

```
>> [X,Y] = meshgrid(1:size(Donut6,1),1:size(Donut6,2));
```

```
>> [Crit, Donut10, FWHM] = MGfit_crit_SPARTA(Donut6, [0 100 50 80 10 4 -45 1.5], 4, 0, 0, 1, X, Y);
```

```
>> Crit
```

```
Crit =
```

```
0
```

```
>> FWHM
```

```
FWHM =
```

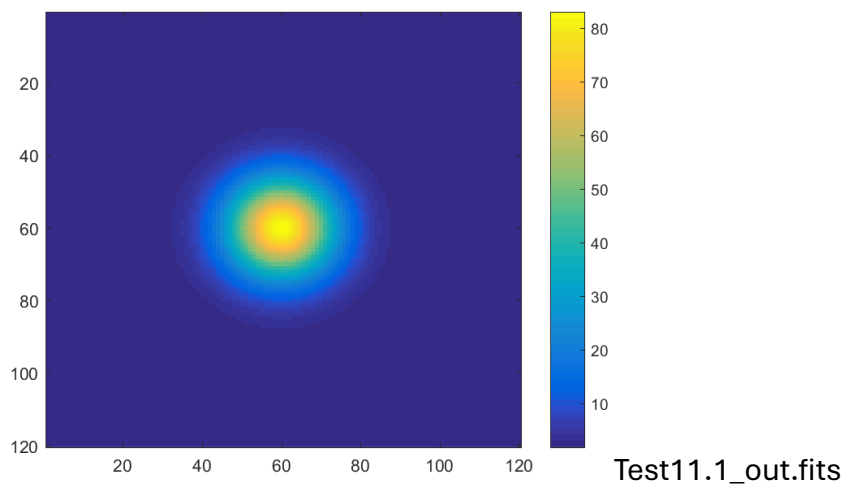
```
15.3284 21.4598
```

## 11. MGfit\_SPARTA

```
% [p_res, fval, psf2, FWHM] = MGfit_SPARTA(psf, fit_meth,  
psf_dl, forced_fit, Cmin, x, y)  
%  
% This function finds the best Gaussian or Moffat fit  
(circular or  
% elliptical) to an input PSF, by minimization of a criterion  
(rms of the  
% difference between the input PSF and a model one) and with  
free  
% parameters the ones describing the model PSF.  
% Custom sub-routines required:  
% 1) First level  
% - MGfit_crit_SPARTA.m  
% 2) Other levels  
% - crop_PSIM.m  
% - conv2.m (Matlab function)
```

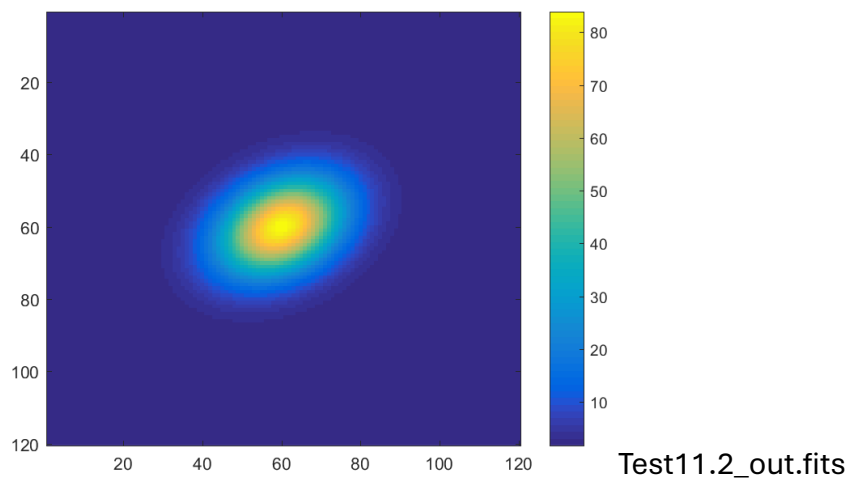
### Test 11.1: Fitting an elongated Moffat with a Circular Gaussian

```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);  
>> [Crit, Donut5, FWHM] = MGfit_crit_SPARTA(Donut, [0 100 60 60 10 4 -45 1.5], 4, 0);  
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(Donut5, 1);  
>> fval(1)  
ans =  
    2.5361  
>> fval(2)  
ans =  
    1.9248  
>> fval(3)  
ans =  
    7.2345e+04  
>> FWHM  
FWHM =  
    22.5014  
>> p_res  
p_res =  
    1.7890 81.2000 60.0000 60.0000 9.5555  
>> figure ; imagesc(psf2) ; colorbar
```



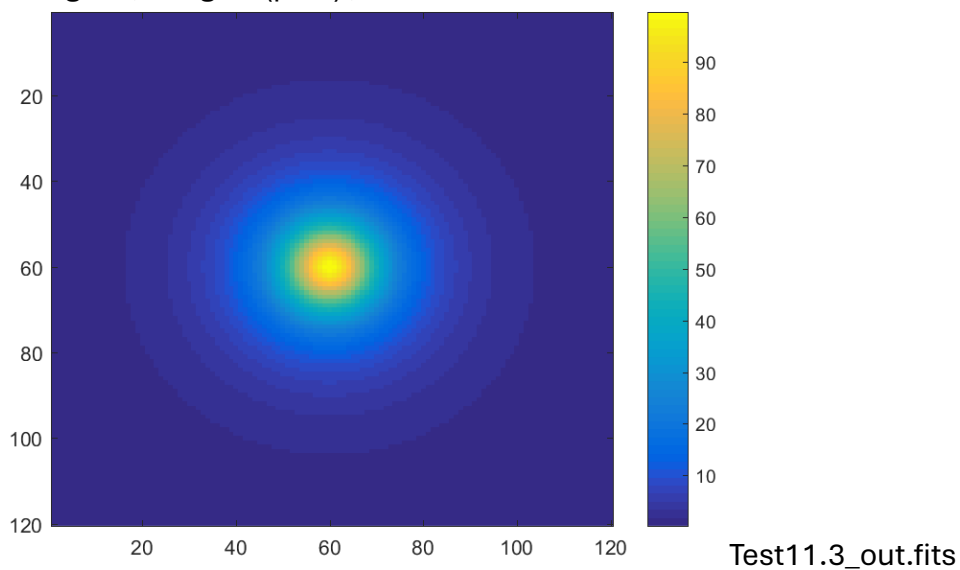
### Test 11.2: Fitting an elongated Moffat with an Elongated Gaussian

```
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(Donut5, 2);
>> fval(1)
ans =
    1.7852
>> fval(2)
ans =
    1.4327
>> fval(3)
ans =
    7.2345e+04
>> FWHM
FWHM =
    19.0537  26.6752
>> p_res
p_res =
    1.7390  82.1385  60.0000  60.0000  8.0914  3.2365  2.1239
>> figure ; imagesc(psf2) ; colorbar
```



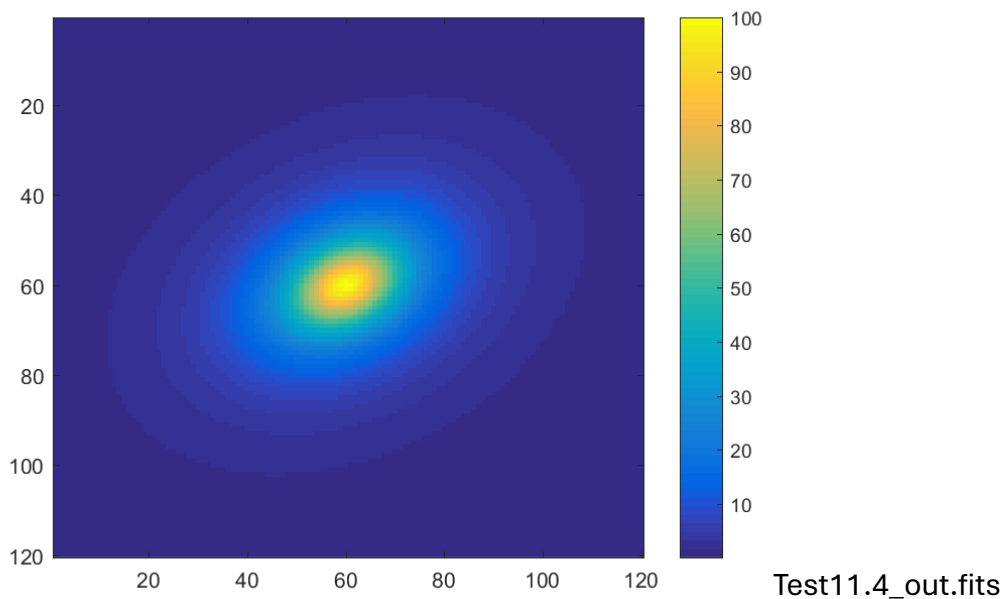
### Test 11.3: Fitting an elongated Moffat with a circular Moffat

```
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(Donut5, 3);  
>> fval(1)  
ans =  
    1.7824  
>> fval(2)  
ans =  
    1.3375  
>> fval(3)  
ans =  
    7.2345e+04  
>> FWHM  
FWHM =  
    17.9497  
>> p_res  
p_res =  
    -0.0424  99.6972  60.0000  60.0000  11.4653  1.4503  
>> figure ; imagesc(psf2) ; colorbar
```



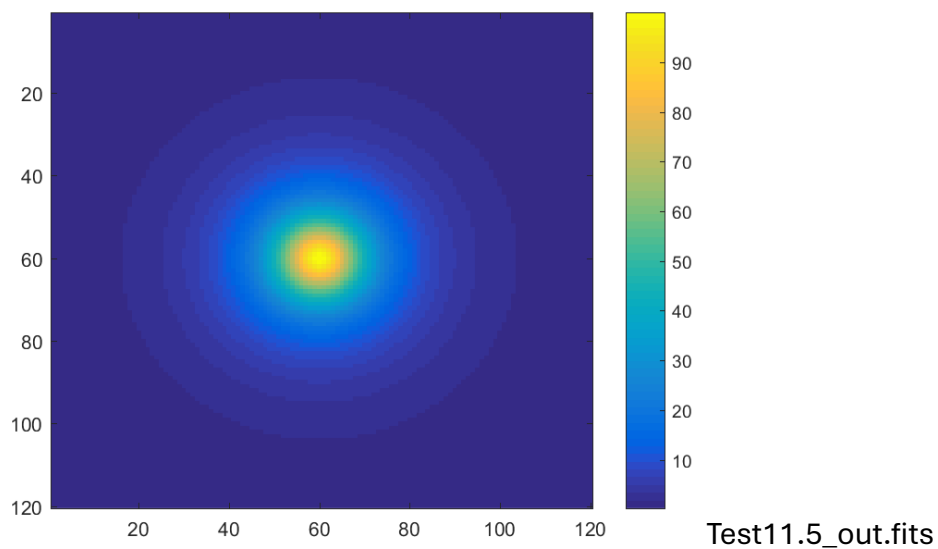
#### Test 11.4: Fitting an elongated Moffat with an elongated Moffat

```
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(Donut5, 4);  
>> fval(1)  
ans =  
    4.7787e-10  
>> fval(2)  
ans =  
    3.5650e-10  
>> fval(3)  
ans =  
    7.2345e+04  
>> FWHM  
FWHM =  
    15.3284    21.4598  
>> p_res  
p_res =  
    0.0000 100.0000 60.0000 60.0000 10.0000 4.0000 2.1239 1.5000  
>> figure ; imagesc(psf2) ; colorbar
```





```
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(Donut5, 3, 0, [0 100 -999 -999 -999 1.45]);
>> fval(1)
ans =
    1.7828
>> fval(2)
ans =
    1.3402
>> fval(3)
ans =
    7.2345e+04
>> FWHM
FWHM =
    17.8815
>> p_res
p_res =
    0 100.0000 60.0000 60.0000 11.4203 1.4500
>> figure ; imagesc(psf2) ; colorbar
```



### Test 11.6: Additional test of centroid\_PSIM with 'MGfit' centroiding methods

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'MGfit', 1); barxy
```

```
barxy =
```

```
0.9045 3.4042
```

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'MGfit', 2); barxy
```

```
barxy =
```

```
0.9547 3.3600
```

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'MGfit', 3); barxy
```

```
barxy =
```

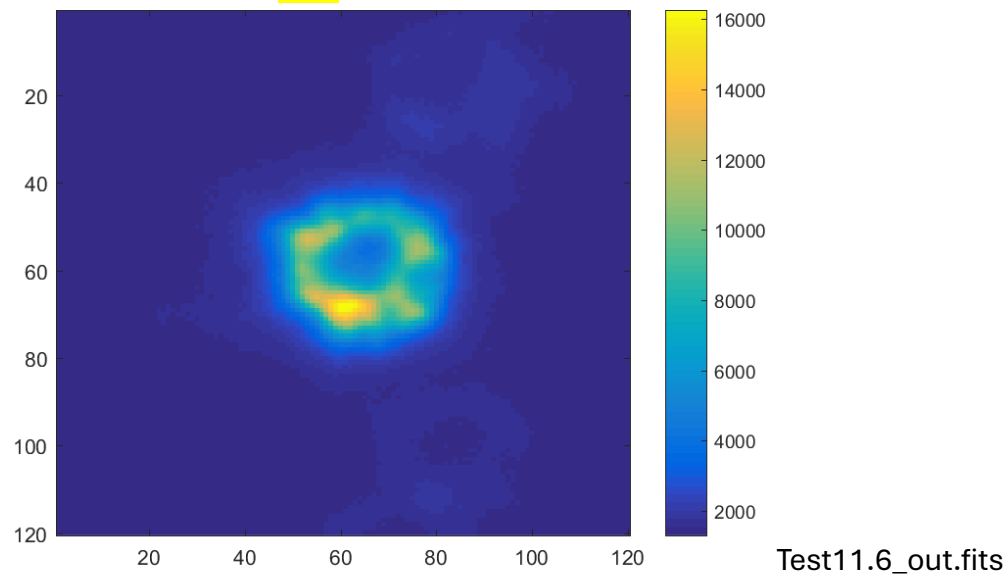
```
4.9081 2.3236
```

```
>> [barxy, psf2] = centroid_PSIM(Donut, 'MGfit', 4); barxy
```

```
barxy =
```

```
4.9447 2.3122
```

```
>> figure ; imagesc(psf2) ; colorbar
```

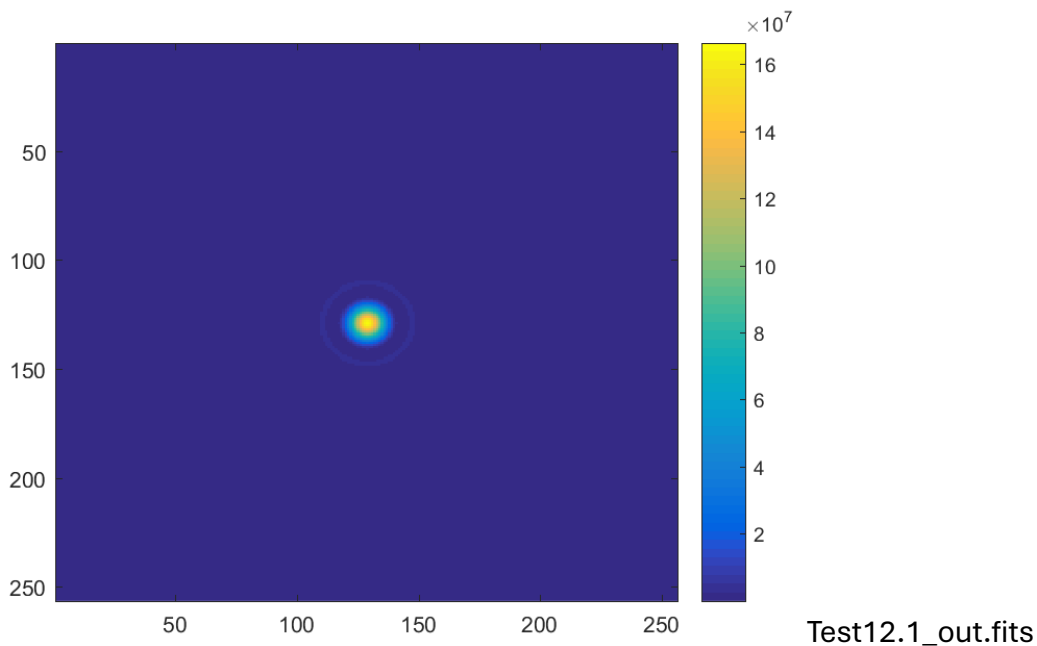


## 12. makepsf\_MIST

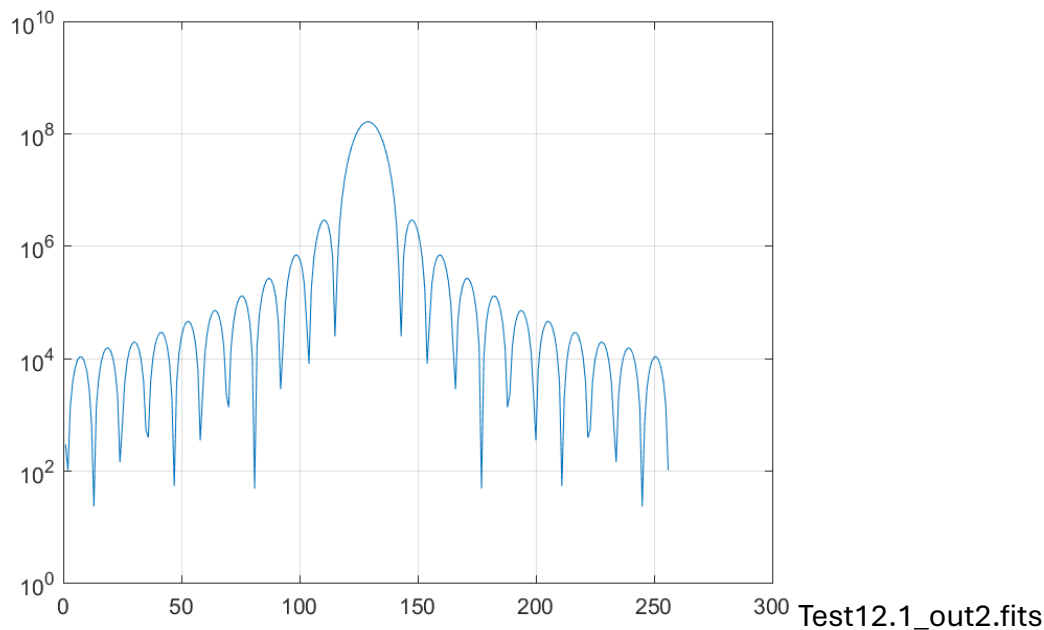
```
% [psf, wl, C, N, zeropad] = makepsf_PSIM(WFmask, pupmask, wl,  
d, pix_scale, nbpixCCD, Srce, Cmin, fit_meth, Tag_bin);  
%  
% This function computes a PSF image from an incoming WF.  
% Custom sub-routines required:  
% 1) First level  
% - bin_PSIM.m  
% - crop_MIST.m  
% - MGfit_crit_SPARTA.m  
% 2) Other levels  
% - centroid_PSIM.m
```

### Test 12.1: Basic PSF, 8-m aperture, 2.2 um

```
>> [Z, in, out] = createZ_PSIM(128, 1, 0, 1);  
>> wl = 2.2 ; d = 8 ; pix_scale = 5/1000 ; nbpixCCD = 256 ;  
>> [psf, wl_out, C, N, zeropad] = makepsf_MIST(Z, Z, wl, d, pix_scale, nbpixCCD, 0, 1);  
>> zeropad  
zeropad =  
    1452  
>> figure ; imagesc(psf) ; colorbar
```



```
>> figure ; semilogy(psf(:,128)) ; grid on
```



### Test 12.2: VISTA parameters / error detection

```
>> [Z, in, out] = createZ_PSIM(128, 1:55, 0.446, 1);
>> wl = 0.73 ; d = 3.7 ; pix_scale = 0.227 ; nbpixCCD = 64 ;
>> [psf, wl_out, C, N, zeropad] = makepsf_MIST(Z(:, :, 1), Z(:, :, 1), wl, d, pix_scale,
nbpixCCD, 0, 1);
```

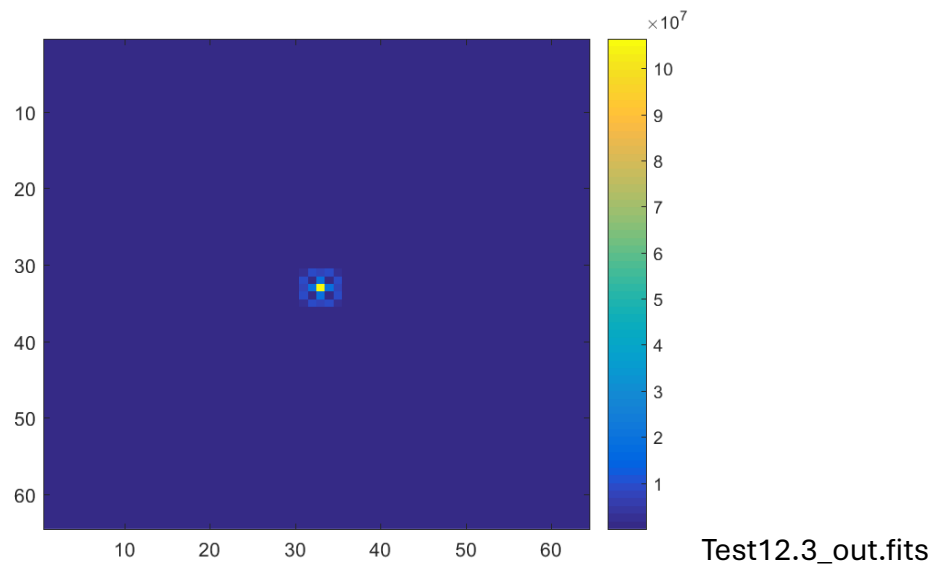
Error using makepsf\_MIST (line 117)

To image the required FoV, the input WF must be at least 714 pixels wide

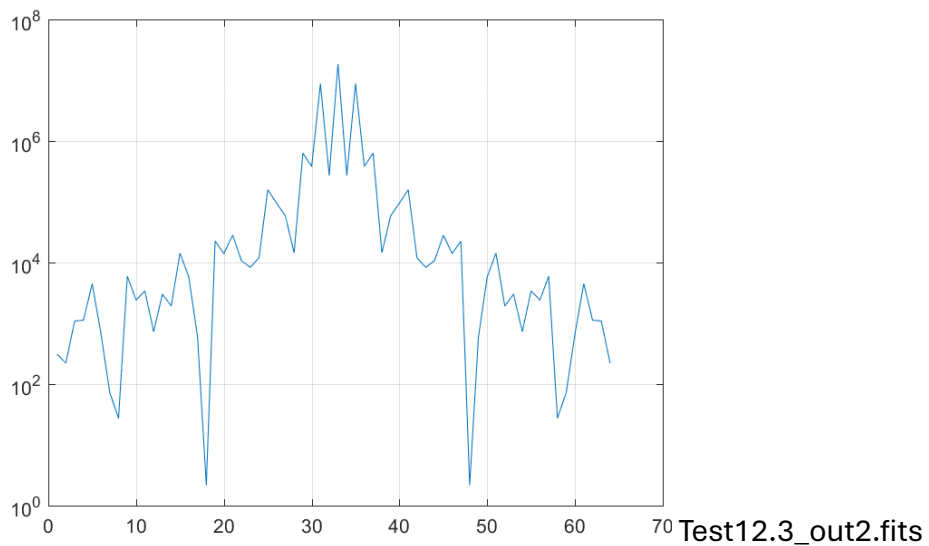
**Imaging the VISTA WFS FoV with the resolution of the Telescope diffraction limit (41 mas at 0.73 um) would require very large Phase maps, but this is not needed, we just need to resolve the seeing, so we play the trick of imaging at 8x the wavelength (=5.84 um) where the diffraction limit is 0.33" (Test 12.3) and convolving the image with a Gaussian seeing blur of a given FWHM (6 pixels in Test 12.4).**

### Test 12.3: Imaging at a longer wavelength

```
>> [psf, wl_out, C, N, zeropad] = makepsf_MIST(Z(:, :, 1), Z(:, :, 1), 8*wl, d, pix_scale,
nbpixCCD, 0, 1);
>> zeropad
zeropad =
    184
>> figure ; imagesc(psf) ; colorbar
```



```
>> figure ; semilogy(psf(:,32)) ; grid on
```



#### Test 12.4: Convolving with a Gaussian Seeing blur

```
[psf, wl_out, C, N, zeropad] = makepsf_MIST(Z(:, :, 1), Z(:, :, 1), 8*wl, d, pix_scale,  
nbpixCCD, 6, 1, 1);  
figure ; imagesc(psf) ; colorbar
```

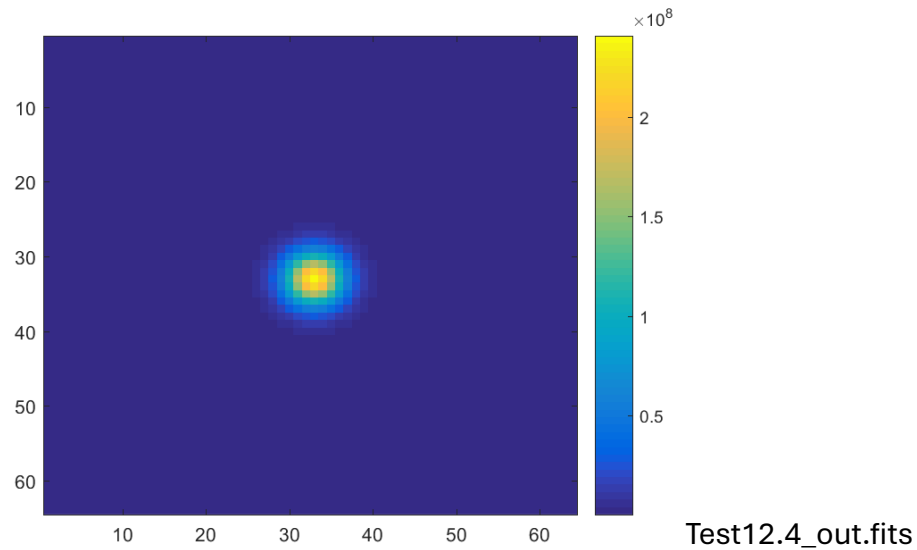
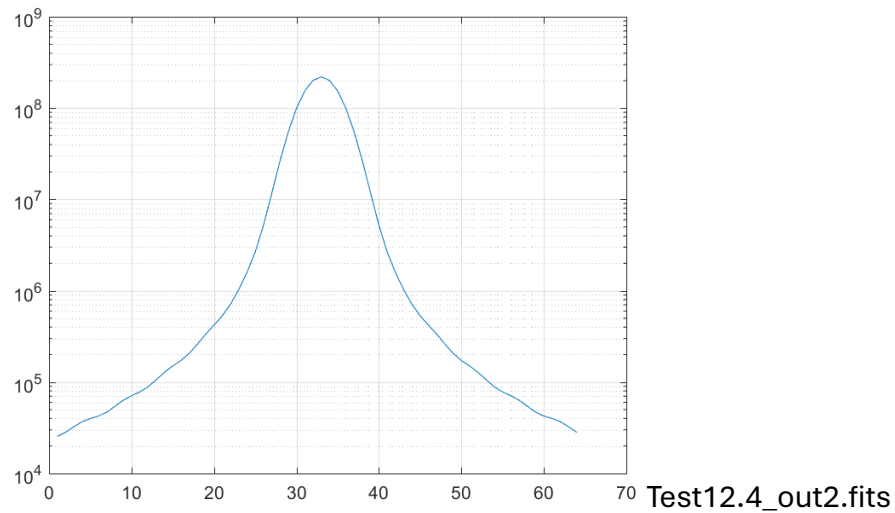


figure ; semilogy(psf(:,32)) ; grid on



**Checking with a Gaussian fit of the PSF center:**

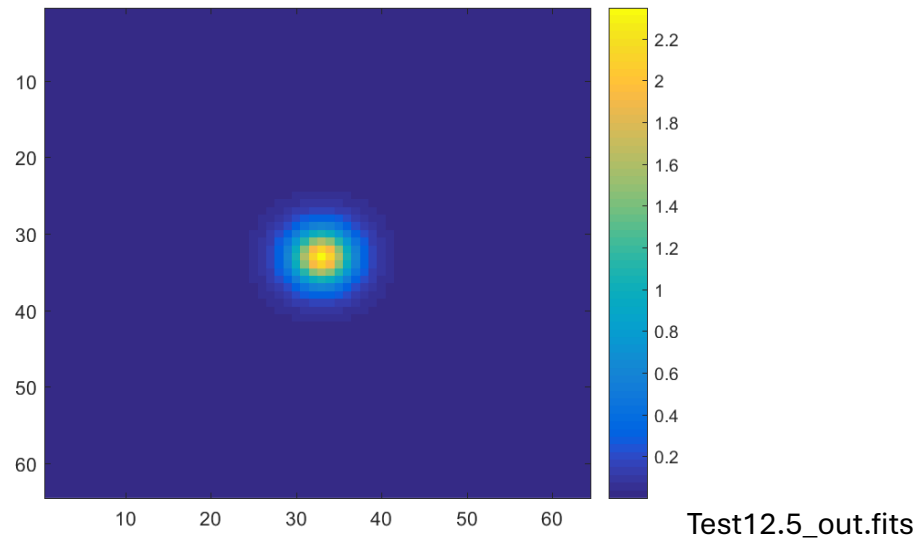
```
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(crop_PSIM(psf*1e-8,32), 1);
>> fval(1)
ans =
    0.0065
>> fval(2)
ans =
    0.0059
>> fval(3)
ans =
    92.8985
>> FWHM
FWHM =
    5.6674
>> p_res
```

```
p_res =
    0.0064    2.3715   17.0000   17.0000    2.4067
```

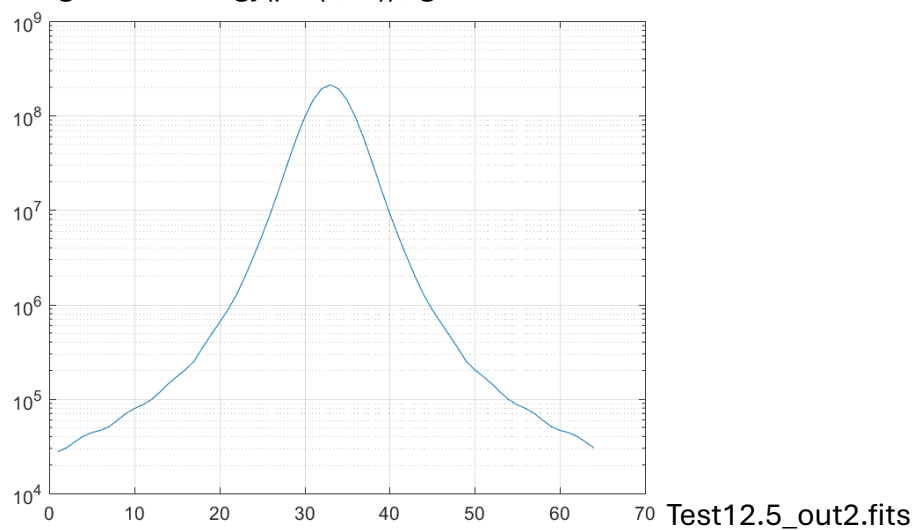
### Test 12.5: Convolving with a Moffat Seeing blur

```
>> [psf, wl_out, C, N, zeropad] = makepsf_MIST(Z(:, :, 1), Z(:, :, 1), 8*wl, d, pix_scale,
nbpixCCD, 6, 1, 3);
```

```
>> figure ; imagesc(psf) ; colorbar
```



```
>> figure ; semilogy(psf(:, 32)) ; grid on
```



### Checking with a Moffat fit of the PSF center:

```
>> [p_res, fval, psf2, FWHM] = MGfit_SPARTA(crop_PSIM(psf*1e-8, 32), 3);
```

```
>> fval(1)
```

```
ans =
```

```
    0.0015
```

```
>> fval(2)
```

```
ans =
```

```
    0.0014
```

```

>> fval(3)
ans =
    99.7745
>> FWHM
FWHM =
    5.4547
>> p_res
p_res =
    0.0028    2.3458   17.0000   17.0000    6.2017    3.9203

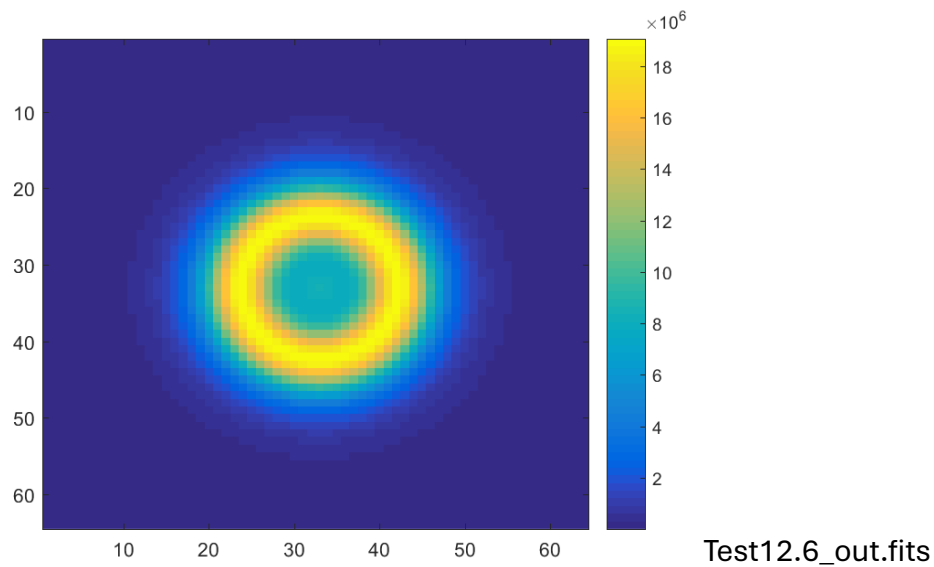
```

### Test 12.6: Defocused PSF

```

>> [psf, wl_out, C, N, zeropad] = makepsf_MIST(4*Z(:,4), Z(:,1), 8*wl, d, pix_scale,
nbpixCCD, 6, 1, 3);
>> figure ; imagesc(psf) ; colorbar

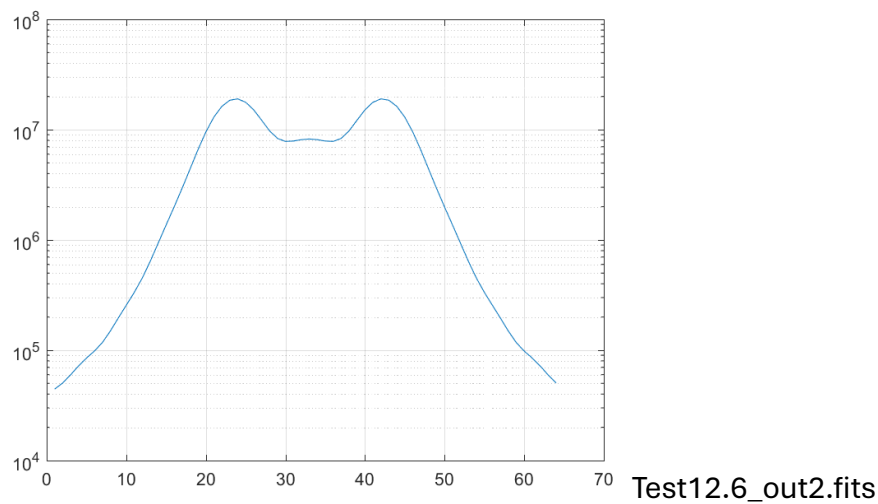
```



```

>> figure ; semilogy(psf(:,32)) ; grid on

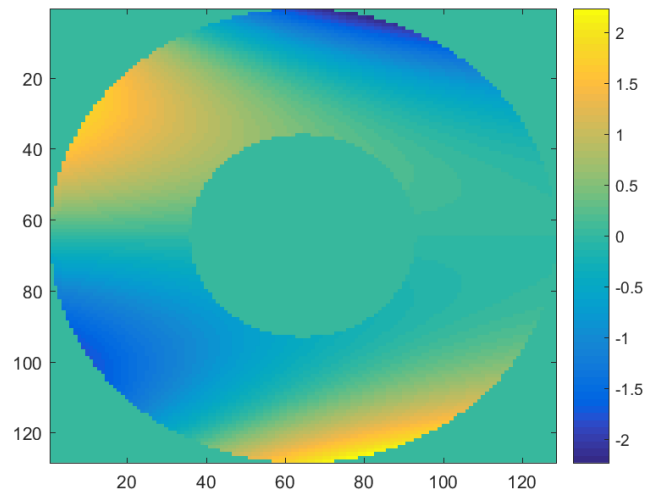
```





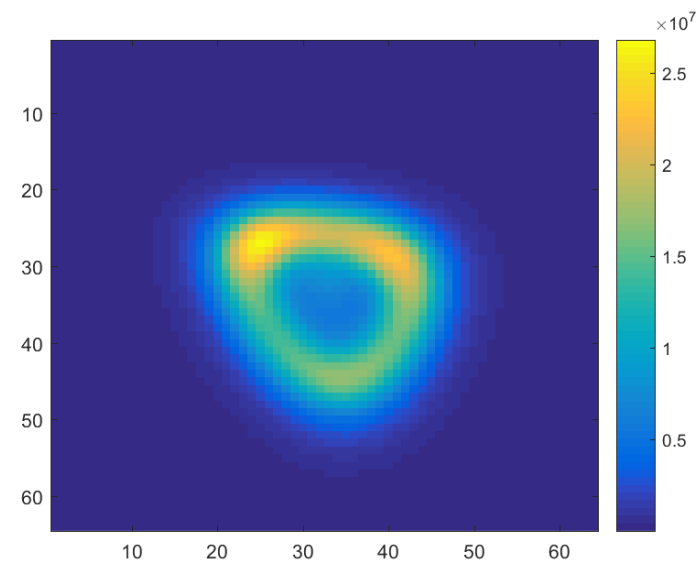
### Test 12.7: Defocused PSF with aberrations

```
>> WF = 0*Z(:,:,1);  
>> Zcoeffs = [0 0 0 0 0.5 0 0.3 0 -0.4 0];  
>> for cpt = 1:10  
>>   WF = WF + Zcoeffs(cpt)*Z(:,:,cpt);  
>> end  
>> figure ; imagesc(WF) ; colorbar
```



Test12.7\_out.fits

```
>> [psf, wl_out, C, N, zeropad] = makepsf_MIST(4*Z(:,:,4)+WF, Z(:,:,1), 8*wl, d, pix_scale,  
nbpixCCD, 6, 1, 1);  
>> figure ; imagesc(psf) ; colorbar
```



Test12.7\_out2.fits

### 13. rotateZ

```
% Zout = rotateZ(Zin, offangle)
%
% This function rotates Zernike coefficients by a given offset
angle.
% Custom sub-routines required:
% - none
```

#### Test 13.1: offangle = 0

```
>> Zin = [0 1 0 3 2 0 3 0 4 0 11 5 0];
>> Zout = rotateZ(Zin, 0);
>> sum(Zout-Zin)
ans =
    6.1232e-17
```

#### Test 13.2: offangle = 45

```
>> Zout = rotateZ(Zin, 45)
Zout =
   -0.7071    0.7071     0    3.0000    0.0000    2.0000    2.1213    2.1213   -2.8284    2.8284
  11.0000    5.0000     0
```

#### Test 13.3: offangle = -90

```
>> Zout = rotateZ(Zin, -90)
Zout =
    1.0000     0     0    3.0000   -2.0000   -0.0000    0.0000   -3.0000   -0.0000    4.0000
  11.0000    5.0000     0
```

#### Test 13.4: input matrix

```
>> Zout = rotateZ([Zin;-Zin], 180)
Zout =
   -0.0000   -1.0000     0    3.0000    2.0000   -0.0000   -3.0000    0.0000   -4.0000    0.0000
  11.0000    5.0000     0
    0.0000    1.0000     0   -3.0000   -2.0000    0.0000    3.0000     0    4.0000    0.0000 -
  11.0000   -5.0000     0
```

## 14. HeadRead

```
% Data = HeadRead(fname, field)
%
% This function extracts the full header or a specific keyword
from a VISTA-4MOST WFS FITS file
% Custom sub-routines required:
% - none
```

### Test 14.1: read full Header

```
>> Data = HeadRead('Test6.1_in.fits')
```

Data =

```
Y: 2025
M: 9
D: 17
h: 0
m: 18
s: 0.8730
DETSIZEX: 2048
DETSIZEY: 2048
TIME: 30
BINX: 1
BINY: 1
NX: 120
NY: 120
STARTX: 218
STARTY: 635
ALT: 70.7630
SEEING: 1.1100
TEMP: 14.5700
MAG1: 12.9000
MAG2: 12.1000
MAG3: 12.4000
MAG4: 12.3000
AZ: 312.8900
GUID_FWHM: 1.9800
ROTANG: -123.8642
M1_TEMP: 16.1100
STR_TEMP: 16.1600
```

### Test 14.2: Extract some values

```
>> Data = HeadRead('Test6.1_in.fits','MAG1')
```

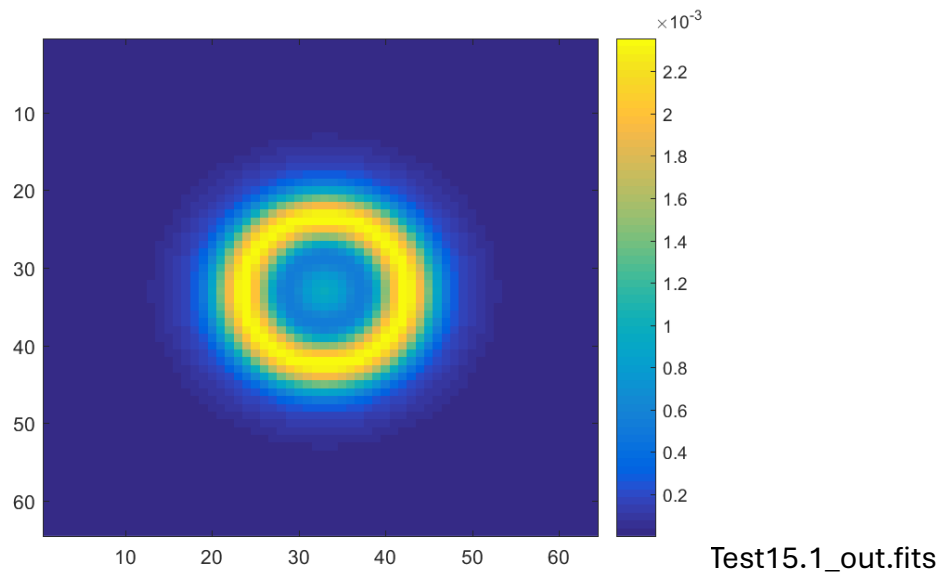
```
Data =  
12.9000  
>> Data = HeadRead('Test6.1_in.fits','s')  
Data =  
0.8730  
>> Data = HeadRead('Test6.1_in.fits','ALT')  
Data =  
70.7630  
>> Data = HeadRead('Test6.1_in.fits','ROTANG')  
Data =  
-123.8642
```

## 15. VISTAfit\_crit

```
% [res, psf_fit] = VISTAfit_crit(psf_in, Z, Zfit_list,  
Zfit_coeffs, Zforce_list, Zforce_coeffs, Pxscale, fit_meth)  
%  
% This function generated a PSF from a list of input Zernike  
coefficients  
% and compares it to an input PSF. It is intended to be used  
as  
% minimization function in VISTAfit.m.  
% Custom sub-routines required:  
% 1) First level  
% - makepsf_MIST.m  
% 2) Other levels  
% - MGfit_crit_SPARTA.m  
% - bin_PSIM.m  
% - crop_MIST.m  
% - centroid_PSIM.m
```

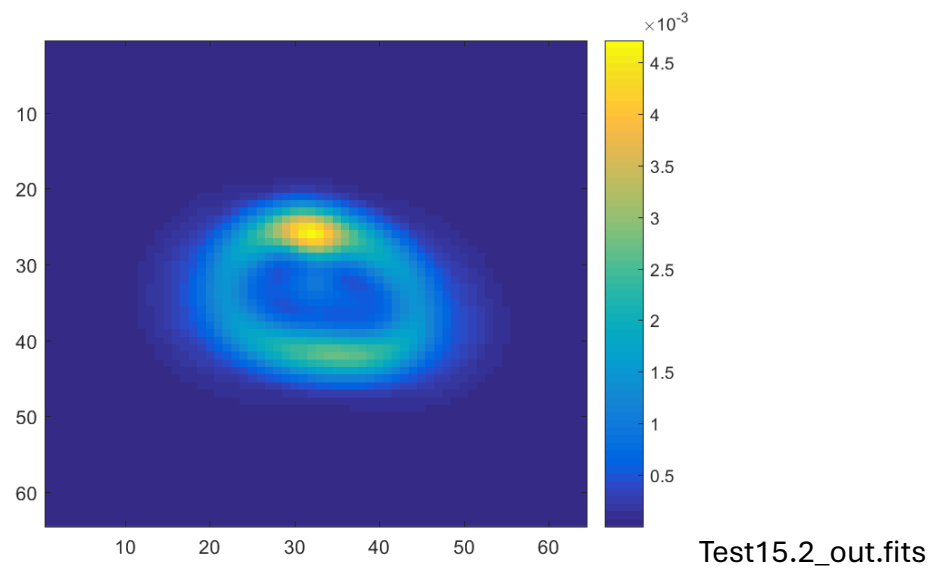
### Test 15.1: simple donut

```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);  
>> [Z, in, out] = createZ_PSIM(128, 1:55, 0.446, 1);  
>> Zfit_coeffs = [4 0 0 4 0 0 0 0 0 0 0 0 0 0 0]+2;  
>> Zfit_list = 0*Zfit_coeffs + 1;  
>> Z2 = reshape(Z(:, :, Zfit_list == 1), size(Z, 1)*size(Z, 2), length(Zfit_list));  
>> Pxscale = 0.227 ; fit_meth = 1; Donut2 = crop_PSIM(Donut, 64, -1) ; Donut2 = Donut2 ./  
sum(Donut2(:));  
>> [res, psf_fit] = VISTAfit_crit(Donut2, Z2, Zfit_list, Zfit_coeffs, 0*Zfit_list, 0*Zfit_list,  
Pxscale, fit_meth);  
>> res  
res =  
0.00040806  
>> figure ; imagesc(psf_fit) ; colorbar
```



**Test 15.2: aberrated donut. Same as Test 15.1 but with:**

```
>> ...
>> Zfit_coeffs = [4 0 0 4 0.5 0 0.3 0 0.2 0 0 0.4 0]+2;
>> ...
>> res
res =
    0.00045398
>> figure ; imagesc(psf_fit) ; colorbar
```



## 16. VISTAfit

```
% [Zfit_coeffs_out, psf_in, psf_fit, T, FWHM, psf_fit_foc,
psf_perf] =...
% VISTAfit(psf_in, Zfit_list, Zfit_coeffs, Zforce_list,
Zforce_coeffs, Obsc, Pxscale)
%
% This function is the "donut Fitter" for the VISTA-4MOST WFS
images. It
% determines the list of Zernike coefficients (plus alignment
defocus) that
% generate a PSF closest to the input one.
% Custom sub-routines required:
% 1) First level
% - createZ_PSIM.m
% - proj.m
% - centroid_PSIM.m
% - fminunc.m (Optimization Toolbox)
% - VISTAfit_crit.m
% - MGfit_SPARTA.m
% - nansum.m (Statistics Toolbox)
% - crop_MIST.m
% 2) Other levels
% - FindNM.m
% - ZPolynomeQuick.m
% - MGfit_crit_SPARTA.m
% - nanmean.m (Statistics Toolbox)
% - makepsf_MIST.m
% - bin_PSIM.m
% - crop_MIST.m
% - centroid_PSIM.m
```

### Test 16.1: Single image processing

```
>> Donut = fitsread('Test6.1_in.fits','Primary',1);
>> Pxscale = 0.227 ; fit_meth = 1; Donut2 = crop_PSIM(Donut,64,-1) ; Donut2 = Donut2 ./
sum(Donut2(:));
>> Zfit_list = zeros(1,105) ; Zfit_list(1:55) = 1; % List of fitted Zernike modes (=1 when
fitted, =0 when not)
>> Zfit_list([11 22 37 56 79]) = 0; % Excluding rotation-symmetric modes from the fit
>> Zfit_coeffs = zeros(1,105); % Starting / target coefficients
>> Zfit_coeffs(1,4) = 4; % initial defocus
% Forced modes (not use in the 2026 code):
>> Zforce_list = zeros(1,105); Zforce_coeffs = zeros(1,105);
>> Zfit_coeffs = (Zfit_coeffs/1000)+2;
>> Zforce_coeffs = (Zforce_coeffs/1000)+2;
```

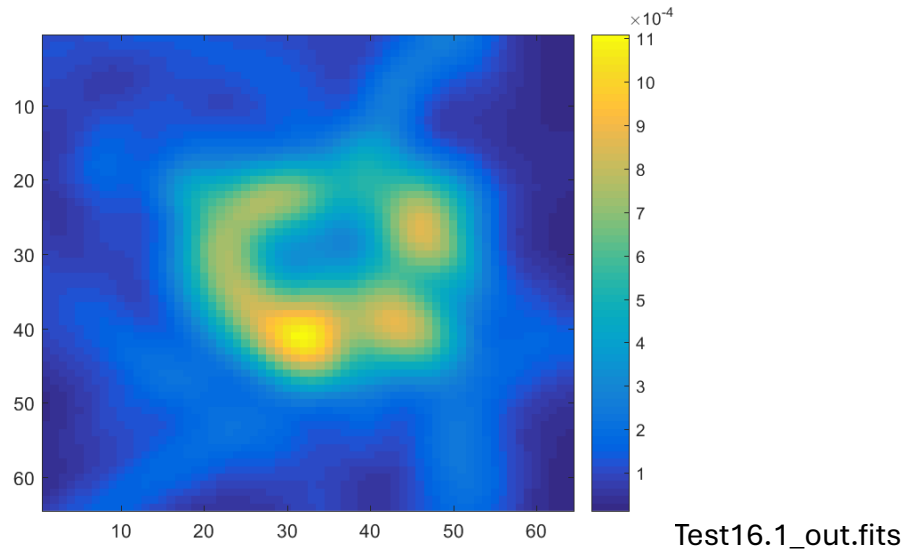
```
>> [Zfit_coeffs_out, psf_in, psf_fit, T, FWHM, psf_fit_foc, psf_perf] =  
VISTAfit(crop_PSIM(Donut,64,-1), Zfit_list, Zfit_coeffs, Zforce_list, Zforce_coeffs, 0.446,  
Pxscale);
```

```
>> FWHM
```

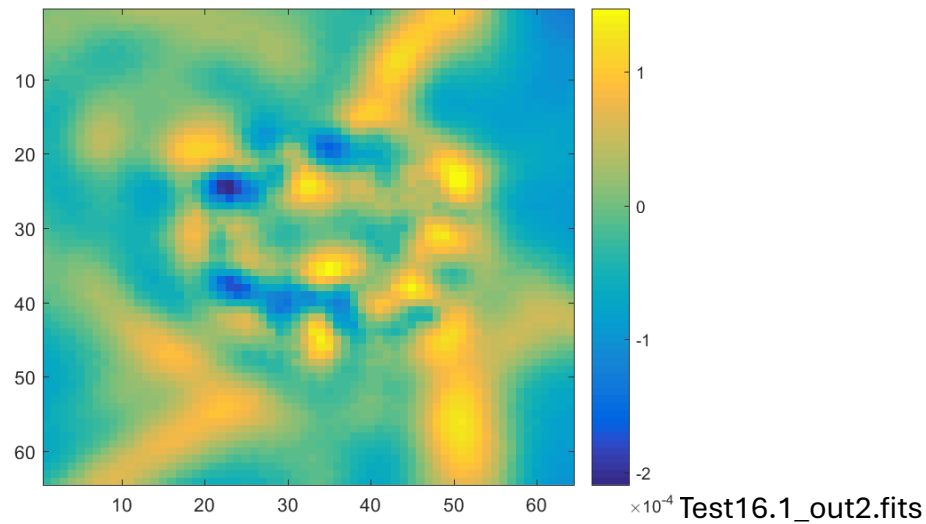
```
FWHM =
```

```
1.2217 0.69225 76.5
```

```
>> figure ; imagesc(psf_fit) ; colorbar
```

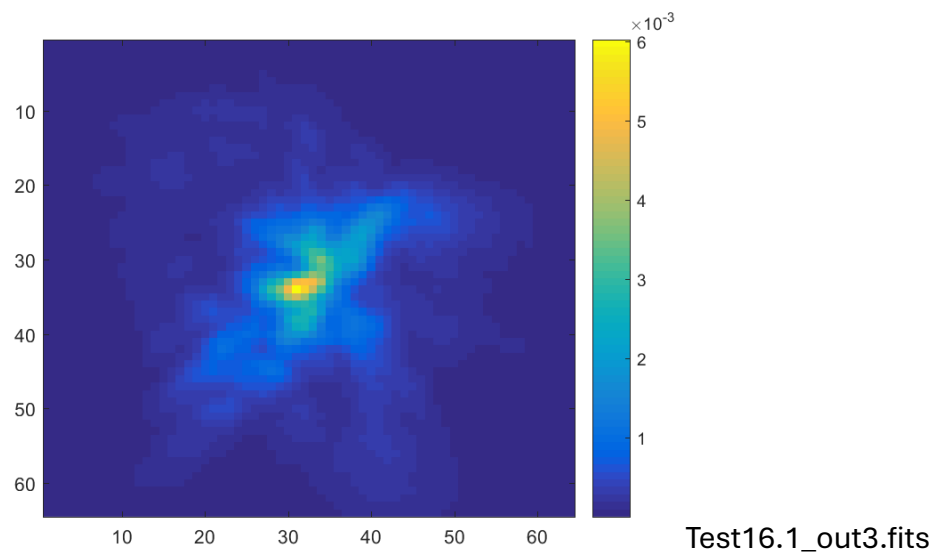


```
>> figure ; imagesc(psf_fit-psf_in) ; colorbar
```

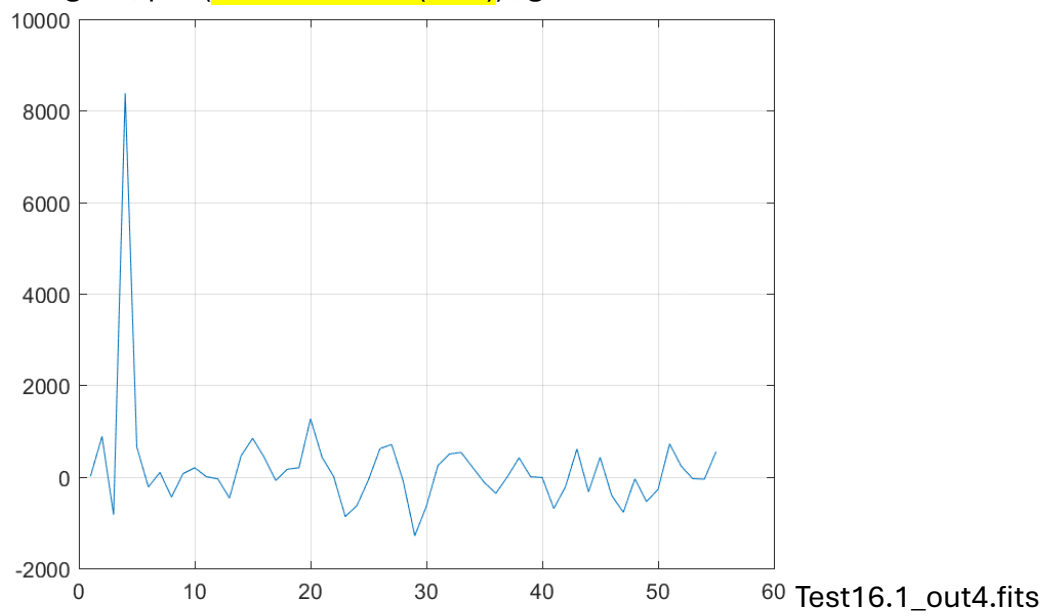


```
>> figure ; imagesc(psf_fit_foc) ; colorbar
```





```
>> figure ; plot(Zfit_coeffs_out(1:55)) ; grid on
```

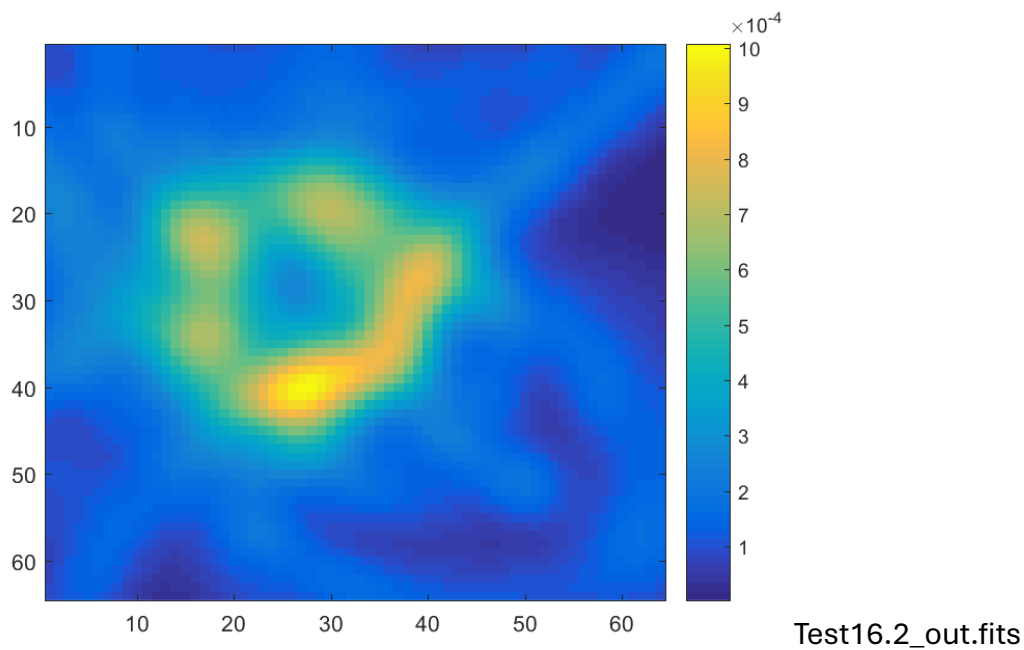


## Test 16.2: Other image processing

### Same test on another image:

```
>> Donut = fitsread('Test16.2_in.fits','Primary',1);  
[...]  
>> FWHM  
FWHM =  
    1.3050    0.6922   88.5000  
>> figure ; imagesc(psf_fit) ; colorbar  
>> figure ; imagesc(psf_fit-psf_in) ; colorbar  
>> figure ; imagesc(psf_fit_foc) ; colorbar  
>> figure ; plot(Zfit_coeffs_out(1:55)) ; grid on
```

### Check:



### And also:

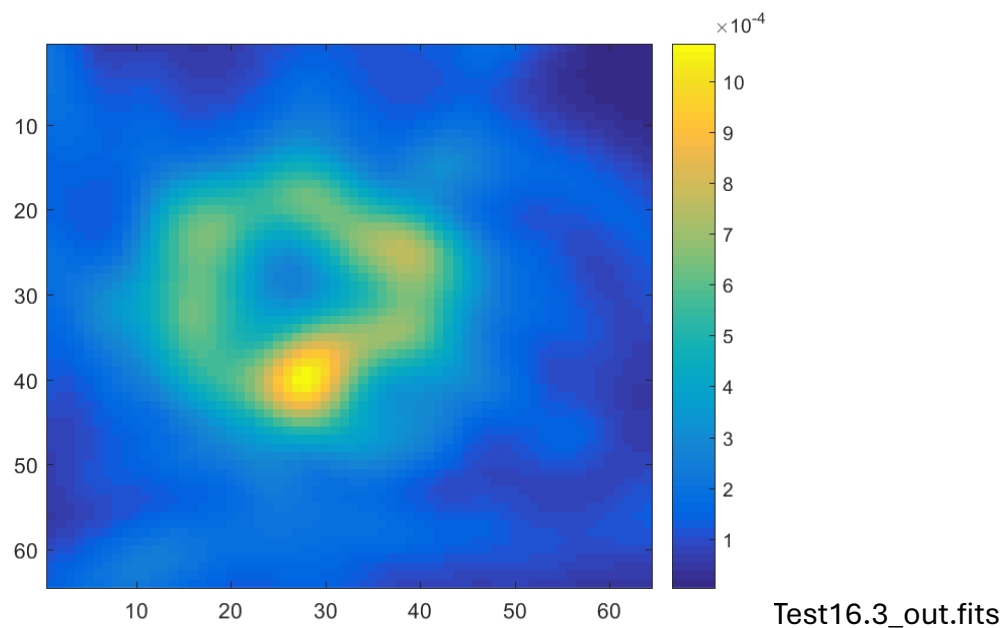
```
Test16.2_out2.fits  
Test16.2_out3.fits  
Test16.2_out4.fits
```

### Test 16.3: sensitivity to input parameters Obsc, Pxscale

Same test as #16.2 but with slightly different parameters

```
>> Donut = fitsread('Test16.2_in.fits','Primary',1);  
>> Pxscale = 0.25 ;  
[...]  
>> [Zfit_coeffs_out, psf_in, psf_fit, T, FWHM, psf_fit_foc, psf_perf] =...  
    VISTAfit(crop_PSIM(Donut,64,-1), Zfit_list, Zfit_coeffs, Zforce_list, Zforce_coeffs, 0.4,  
Pxscale);  
>> FWHM  
FWHM =  
    1.7470  0.6449 170.9000  
>> figure ; imagesc(psf_fit) ; colorbar  
>> figure ; imagesc(psf_fit-psf_in) ; colorbar  
>> figure ; imagesc(psf_fit_foc) ; colorbar  
>> figure ; plot(Zfit_coeffs_out(1:55)) ; grid on
```

**Check:**



**And also:**

Test16.3\_out2.fits

Test16.3\_out3.fits

Test16.3\_out4.fits

### Test 16.4: sensitivity to number of fitted Zernikes

Same test as #16.2 but less fitted modes:

[...]

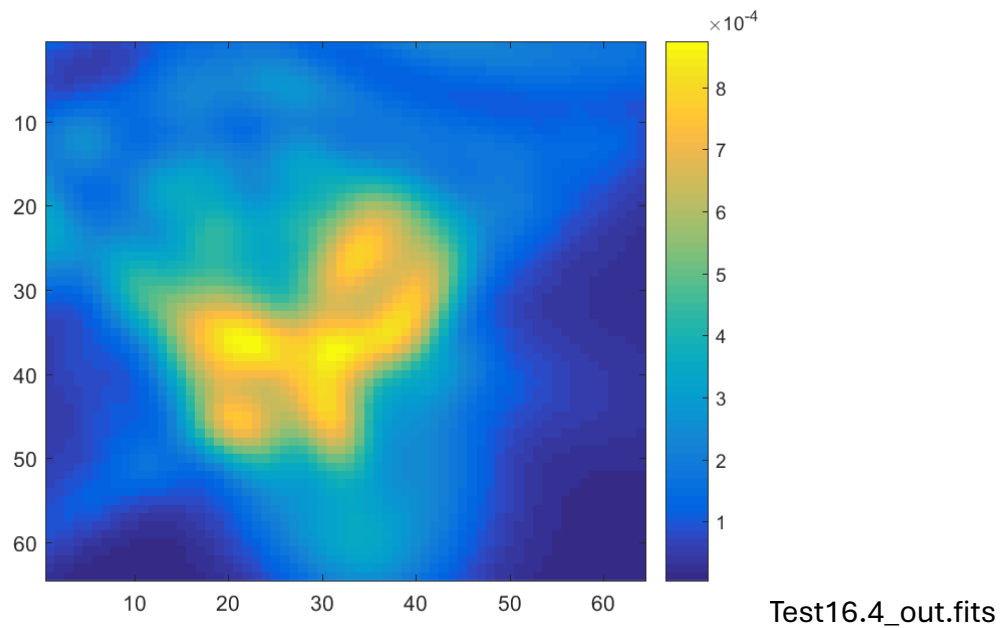
```
>> Zfit_list = zeros(1,105) ; Zfit_list(1:31) = 1;
```

[...]

FWHM =

3.7610 0.6922 443.3000

**Check:**



**And also:**

Test16.4\_out2.fits

Test16.4\_out3.fits

Test16.4\_out4.fits

## 17. MIST\_Execute

```
% [donutFitter_Out, MIST_Out, error_Out] =  
MIST_Execute(config, calibration, donutFitter_In)  
%  
% This function computes the delta commands to be applied to  
the VISTA  
% Telescope M1 and M2 adaptive mirrors to correct for the  
aberrations  
% estimated in the input defocused images.  
% Custom sub-routines required:  
% 1) First level  
% - HeadRead.m  
% - VISTAfit.m  
% - rotateZ.m  
% - bin_PSIM.m  
% - crop_MIST.m  
% 2) Other levels  
% - centroid_PSIM.m  
% - VISTAfit_crit.m  
% - createZ_PSIM.m  
% - FindNM.m  
% - ZPolynomeQuick.m  
% - proj.m  
% - fminunc.m (Optimization Toolbox)  
% - MGfit_SPARTA.m  
% - MGfit_crit_SPARTA.m  
% - nansum.m (Statistics Toolbox)  
% - nanmean.m (Statistics Toolbox)  
% - makepsf_MIST.m
```

**The full MIST\_Execute routine can be tested on different sets of images. First, there are 3 different ways of generating the input structures for the routine (which need to be translated to the Python equivalent).**

**Test 17.1: test MIST\_Execute directly with the TCS, which we have verified on 02/03/26 uses the same inputs as the tests 17.2 and 17.3 below.**

**Test 17.2: Load the structures saved in the provided mat file Test17.2\_in.mat**  
>> load Test17.2\_in.mat

### Test 17.3: generate the structures by hand

```
>> Input_path = 'C:\[aaa]\'; % Location of input matrices
>> baseDir = 'C:\[bbb]\'; cd(baseDir) % Location of data
>> clear calibration config donutFitter_In

>> calibration.controlMatrixPath = [Input_path, 'vtmdatControlMatrix.mat']; % Path for
the Control Matrix
>> calibration.ncpaPath = [Input_path, 'vtmdatNCPA.mat']; % Path for the field
aberrations vectors
>> calibration.zP = [1.4879e+06, 1.3685e+06, 8.4221e+05, 7.5807e+05]; % Zero Point
Flux (ADUs)
>> calibration.zPM = [13.83 13.83 14.63 14.63]; % Zero Point Magnitude
>> calibration.zPX = [20 20 20 20]; % Zero Point Exposure time (s)

>> config.verbosityLevel = 1; % Verbosity Level
>> config.logLevel = 1; % Log Level
>> config.plotDonuts = 1; % (0 or 1) for plotting the input, output and difference donuts
>> config.defoc = [4000 -4000 4000 -4000]; % Alignment defocus (nm rms) for each
donut
>> config.nbpixLO = 64; % Linear number of pixels in the images (LO mode)
>> config.nbpixHO = 64; % Linear number of pixels in the images (HO mode)
>> config.pxscale = 0.227; % Pixel scale (arcsec / px)
>> config.obsc = 0.446; % Ratio of central obscuration (relative to M1 diameter)

>> donutFitter_In.zfit_list = zeros(1,105) ;
>> donutFitter_In.zfit_list(1:55) = 1; % List of fitted Zernike modes (=1 when fitted, =0
when not)
>> donutFitter_In.zfit_list([11 22 37 56 79]) = 0; % Excluding rotation-symmetric modes
from the fit
>> donutFitter_In.fittedDonutImagesDir = [baseDir, 'MIST_out\']; % Location of output
data
>> donutFitter_In.wfs1FilePath = 'VISTA_TCCDWFS1_2025-09-
17T00%3A18%3A00.967.fits'
>> donutFitter_In.wfs2FilePath = 'VISTA_TCCDWFS2_2025-09-
17T00%3A18%3A00.873.fits'
>> donutFitter_In.wfs3FilePath = 'VISTA_TCCDWFS3_2025-09-
17T00%3A18%3A00.869.fits'
>> donutFitter_In.wfs4FilePath = 'VISTA_TCCDWFS4_2025-09-
17T00%3A18%3A00.968.fits'
```

#### Test 17.4: check the input structures

```
>> calibration
```

```
calibration =
```

```
    controlMatrixPath: 'C:\[aaa]\vtmdatControlMatrix.mat'
```

```
        ncpaPath: 'C:\[aaa]\vtmdatNCPA.mat'
```

```
        zP: [1487900 1368500 842210 758070]
```

```
        zPM: [13.8300 13.8300 14.6300 14.6300]
```

```
        zPX: [20 20 20 20]
```

```
>> config
```

```
config =
```

```
    verbosityLevel: 1
```

```
        logLevel: 1
```

```
    plotDonuts: 1
```

```
        defoc: [4x1 double]
```

```
        nbpixLO: 64
```

```
        nbpixHO: 64
```

```
        pxscale: 0.2270
```

```
        obsc: 0.4460
```

```
>> config.defoc
```

```
ans =
```

```
    4000
```

```
   -4000
```

```
    4000
```

```
   -4000
```

```
>> donutFitter_In
```

```
donutFitter_In =
```

```
    zfit_list: [1x105 double]
```

```
    fittedDonutImagesDir: 'C:\[bbb]\MIST_out\'
```

```
        wfs1FilePath: 'VISTA_TCCDWFS1_2025-09-17T00%3A18%3A00.967.fits'
```

```
        wfs2FilePath: 'VISTA_TCCDWFS2_2025-09-17T00%3A18%3A00.873.fits'
```

```
        wfs3FilePath: 'VISTA_TCCDWFS3_2025-09-17T00%3A18%3A00.869.fits'
```

```
        wfs4FilePath: 'VISTA_TCCDWFS4_2025-09-17T00%3A18%3A00.968.fits'
```

### Test 17.5: run Execute\_MIST and check the displayed output

```
>> [donutFitter_Out, MIST_Out, error_Out] = MIST_Execute(config, calibration,  
donutFitter_In);
```

Starting MIST\_Execute...

The verbosityLevel is 1

The logLevel is 1

The donuts will be plotted

There are 4 input images

CM file loaded from C:\[aaa]\vtmdatControlMatrix.mat

NCPA file loaded from C:\[aaa]\vtmdatNCPA.mat

Loading images...

There are 4 valid images

Setting is LOWFS

Processing images...

Start fitting image#1...

Completed fitting image#1.

Start fitting image#2...

Completed fitting image#2.

Start fitting image#3...

Completed fitting image#3.

Start fitting image#4...

Completed fitting image#4.

There are 4 valid fits

Display of the Donuts...

MIST: Estimated Magnitudes: {13.2, 12.4, 12.7, 13.1}

Generating donutFitter\_Out outputs...

Saving images...

MIST: Trying to write file to directory C:\[bbb]\MIST\_out\

Saving images...

MIST: Trying to write file to directory C:\[bbb]\MIST\_out\

Saving images...

MIST: Trying to write file to directory C:\[bbb]\MIST\_out\

Saving images...

MIST: Trying to write file to directory C:\[bbb]\MIST\_out\

MIST: WFS#1 Zernike modes 4-10 (nm rms): {-1542, -1, 62, -88, 22, -318, -313}

MIST: WFS#1 Rotated Zernike modes 4-10 (nm rms): {-1542, 57, -25, 30, -86, -267, -357}

MIST: WFS#2 Zernike modes 4-10 (nm rms): {-1171, -97, -112, -421, 65, 41, -76}

MIST: WFS#2 Rotated Zernike modes 4-10 (nm rms): {-1171, -81, -124, 424, -36, -55, 67}

MIST: WFS#3 Zernike modes 4-10 (nm rms): {-775, -452, 37, 177, 38, -423, -129}



MIST: WFS#3 Rotated Zernike modes 4-10 (nm rms): {-775, 198, 408, 131, -125, 383, 222}

MIST: WFS#4 Zernike modes 4-10 (nm rms): {-846, -154, 101, -376, -249, -110, 164}

MIST: WFS#4 Rotated Zernike modes 4-10 (nm rms): {-846, -167, 80, -358, -274, -141, 139}

MIST: Estimated FWHM from fit (arcsec)= {1.27, 1.51, 1.34, 1.46}

MIST: Estimated FWHM from aberrations (arcsec)= {0.85, 0.84, 1.27, 0.81}

MIST: Estimated FWHM enlargement due to aberrations (percent) = {22.8, 21.3, 83.4, 17.7}

Generating TSIM\_Out outputs...

MIST: Corrections to M1 B21c, B21s, B31c, B31s (nm rms): {42, 1, 20, 18}

MIST: Corrections to M2 Z4-Z8 (nm rms): {759, -71, -43, 65, 28}

MIST: Total Residual WF (nm rms): {1513}

Generating error\_Out outputs...

MIST\_Execute completed.

### Test 17.6: check the output structures

```
>> donutFitter_Out
```

```
donutFitter_Out =
```

```
    rawZern: [4x105 double]
    rotatedZern: [4x105 double]
        time: [4x1 double]
    magnitude: [4x1 double]
        fwhm: [4x4 double]
        snr: [4x4 double]
totalResidual: 1.5132e+03
fittedDonutImagesPath: {4x1 cell}
        fitQ: [4x1 double]
        hdr: [4x1 struct]
    psf_fit_foc: [64x64x4 double]
    psf_perf: [64x64x4 double]
```

```
>> donutFitter_Out.magnitude
```

```
ans =
```

```
13.2070
12.3590
12.7434
13.0804
```

```
>> donutFitter_Out.fwhm
```

```
ans =
```

```
1.2743 0.8498 0.6922 22.8000
1.5068 0.8398 0.6922 21.3000
1.3448 1.2696 0.6922 83.4000
1.4587 0.8147 0.6922 17.7000
```

```
>> donutFitter_Out.snr(:,[1 3])
```

```
ans =
```

```
1.0e+06 *
3.9616 0.4584
7.9571 0.5754
7.1808 0.5204
4.7384 0.0434
```

```
>> donutFitter_Out.snr(:,[2 4])
```

```
ans =
```

```
1.0e+03 *
0.0086 1.9171
0.0138 1.0768
0.0138 2.0148
0.1091 0.0942
```

```
>> donutFitter_Out.fittedDonutImagesPath
ans =
'VISTA_MIST_CWFS#1_[processing timestamp].fits'
'VISTA_MIST_CWFS#2_[processing timestamp].fits'
'VISTA_MIST_CWFS#3_[processing timestamp].fits'
'VISTA_MIST_CWFS#4_[processing timestamp].fits'
```

```
>> donutFitter_Out.fitQ
```

```
ans =
144.1699
404.1214
325.5070
261.6003
```

```
>> donutFitter_Out.hdr(1)
```

```
ans =
    Y: 2025
    M: 9
    D: 17
    h: 0
    m: 18
    s: 0.9675
DETSIZEX: 2048
DETSIZEY: 2048
    TIME: 30
    BINX: 1
    BINY: 1
    NX: 120
    NY: 120
STARTX: 82
STARTY: 1693
    ALT: 70.7630
SEEING: 1.1100
    TEMP: 14.5700
    MAG1: 12.9000
    MAG2: 12.1000
    MAG3: 12.4000
    MAG4: 12.3000
    AZ: 312.8900
GUID_FWHM: 1.9800
    ROTANG: -123.8642
    M1_TEMP: 16.1100
    STR_TEMP: 16.1600
```

```
>> MIST_Out
```

```
MIST_Out =
```

```
  m2commands: [759 -71 -43 65 28]
```

```
  m1commands: [42 1 20 18 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
>> error_Out
```

```
error_Out =
```

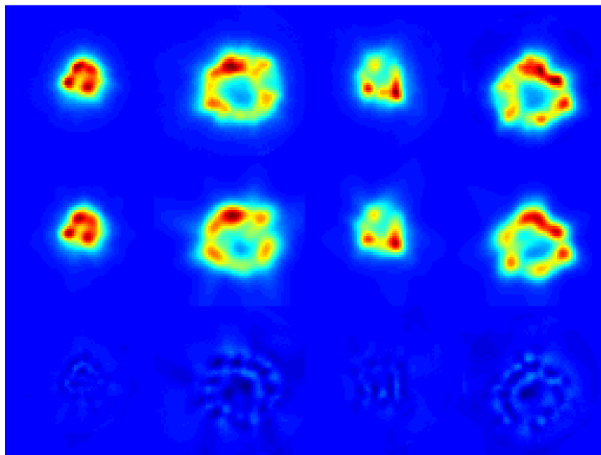
```
  errorStatus: 0
```

```
  errorStack: {10x1 cell}
```

### Test 17.7: check the output images

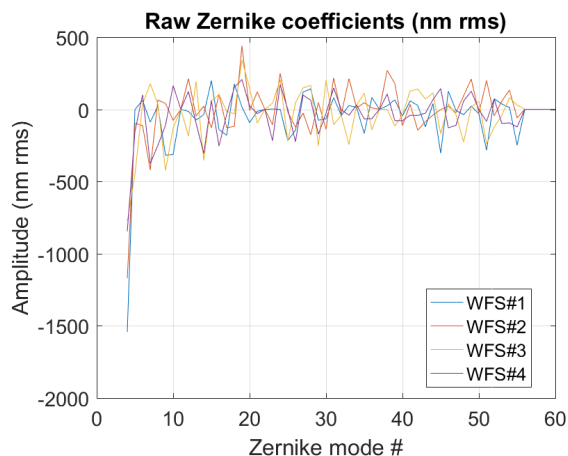
% Display of images in the right order for Matlab:

```
>> for cpt = 1:4
>> psf_all(:, :, cpt) = fitsread(['MIST_Out/VISTA_MIST_CWFS#', num2str(cpt),
'_[processing timestamp].fits']);
>> end
>> figure('Position',[10 10 900 700])
>> imagesc(flipud([psf_all(:, :, 1)./max(max(psf_all(:, :, 1))) ...
psf_all(:, :, 2)./max(max(psf_all(:, :, 2))) ...
psf_all(:, :, 3)./max(max(psf_all(:, :, 3))) ...
psf_all(:, :, 4)./max(max(psf_all(:, :, 4)))]))
>> colorbar off ; axis normal equal tight off ; colormap('jet') ;
```



Test17.4\_out1.png

```
>> figure ; plot(4:60, donutFitter_Out.rawZern(:, 4:60)) ; grid on
>> title('Raw Zernike coefficients (nm rms)', 'FontSize', 20)
>> xlabel('Zernike mode #', 'FontSize', 14)
>> ylabel('Amplitude (nm rms)', 'FontSize', 14)
>> set(gca, 'FontSize', 14)
>> legend('WFS#1', 'WFS#2', 'WFS#3', 'WFS#4', 'location', 'best')
```



Test17.4\_out2.png

**Alternatively, load the Zernike coefficients from the header of one of the output image files:**

```
>> rawZern = fitsread('MIST_Out/VISTA_MIST_CWFS#1_[processing  
timestamp].fits','Image',1);  
>> figure ; plot(4:60,rawZern(:,4:60)) ; grid on  
>> title('Raw Zernike coefficients (nm rms)','FontSize',20)  
>> xlabel('Zernike mode #','FontSize',14)  
>> ylabel('Amplitude (nm rms)','FontSize',14)  
>> set(gca,'FontSize',14)  
>> legend('WFS#1','WFS#2','WFS#3','WFS#4','location','best')
```

**Test 17.8: Full run on several test sets. The following test data is provided:**

VISTA\_TCCDWFS1\_2025-09-17T00%3A18%3A00.967.fits  
VISTA\_TCCDWFS1\_2025-09-17T00%3A19%3A20.280.fits  
VISTA\_TCCDWFS1\_2025-09-17T00%3A20%3A37.547.fits  
VISTA\_TCCDWFS1\_2025-09-17T00%3A22%3A56.062.fits  
VISTA\_TCCDWFS1\_2025-09-17T00%3A26%3A48.557.fits  
VISTA\_TCCDWFS1\_2025-09-17T00%3A28%3A15.302.fits  
VISTA\_TCCDWFS2\_2025-09-17T00%3A18%3A00.873.fits  
VISTA\_TCCDWFS2\_2025-09-17T00%3A19%3A20.183.fits  
VISTA\_TCCDWFS2\_2025-09-17T00%3A20%3A37.445.fits  
VISTA\_TCCDWFS2\_2025-09-17T00%3A22%3A56.164.fits  
VISTA\_TCCDWFS2\_2025-09-17T00%3A26%3A48.609.fits  
VISTA\_TCCDWFS2\_2025-09-17T00%3A28%3A15.352.fits  
VISTA\_TCCDWFS3\_2025-09-17T00%3A18%3A00.869.fits  
VISTA\_TCCDWFS3\_2025-09-17T00%3A19%3A20.178.fits  
VISTA\_TCCDWFS3\_2025-09-17T00%3A20%3A37.446.fits  
VISTA\_TCCDWFS3\_2025-09-17T00%3A22%3A56.064.fits  
VISTA\_TCCDWFS3\_2025-09-17T00%3A26%3A48.457.fits  
VISTA\_TCCDWFS3\_2025-09-17T00%3A28%3A15.204.fits  
VISTA\_TCCDWFS4\_2025-09-17T00%3A18%3A00.968.fits  
VISTA\_TCCDWFS4\_2025-09-17T00%3A19%3A20.280.fits  
VISTA\_TCCDWFS4\_2025-09-17T00%3A20%3A37.551.fits  
VISTA\_TCCDWFS4\_2025-09-17T00%3A22%3A56.161.fits  
VISTA\_TCCDWFS4\_2025-09-17T00%3A26%3A48.614.fits  
VISTA\_TCCDWFS4\_2025-09-17T00%3A28%3A15.351.fits

**One should run Execute\_MIST on each set of 4 images with a similar timestamp, and compare the results with the ones produced by the current Matlab code, also provided in the folder MIST\_Out. They were analysed in the chronological order so there is a 1:1 match between the list above and the one below:**

VISTA\_MIST\_CWFS#1\_2026-03-10T11-39-04.fits  
VISTA\_MIST\_CWFS#1\_2026-03-10T12-05-34.fits  
VISTA\_MIST\_CWFS#1\_2026-03-10T12-10-27.fits  
VISTA\_MIST\_CWFS#1\_2026-03-10T12-22-44.fits  
VISTA\_MIST\_CWFS#1\_2026-03-10T12-24-57.fits  
VISTA\_MIST\_CWFS#1\_2026-03-10T12-27-01.fits  
VISTA\_MIST\_CWFS#2\_2026-03-10T11-39-04.fits  
VISTA\_MIST\_CWFS#2\_2026-03-10T12-05-34.fits  
VISTA\_MIST\_CWFS#2\_2026-03-10T12-10-27.fits  
VISTA\_MIST\_CWFS#2\_2026-03-10T12-22-44.fits  
VISTA\_MIST\_CWFS#2\_2026-03-10T12-24-57.fits

VISTA\_MIST\_CWFS#2\_2026-03-10T12-27-01.fits  
 VISTA\_MIST\_CWFS#3\_2026-03-10T11-39-04.fits  
 VISTA\_MIST\_CWFS#3\_2026-03-10T12-05-34.fits  
 VISTA\_MIST\_CWFS#3\_2026-03-10T12-10-27.fits  
 VISTA\_MIST\_CWFS#3\_2026-03-10T12-22-45.fits  
 VISTA\_MIST\_CWFS#3\_2026-03-10T12-24-57.fits  
 VISTA\_MIST\_CWFS#3\_2026-03-10T12-27-01.fits  
 VISTA\_MIST\_CWFS#4\_2026-03-10T11-39-04.fits  
 VISTA\_MIST\_CWFS#4\_2026-03-10T12-05-34.fits  
 VISTA\_MIST\_CWFS#4\_2026-03-10T12-10-27.fits  
 VISTA\_MIST\_CWFS#4\_2026-03-10T12-22-45.fits  
 VISTA\_MIST\_CWFS#4\_2026-03-10T12-24-57.fits  
 VISTA\_MIST\_CWFS#4\_2026-03-10T12-27-01.fits

**One should compare between the Matlab and Python results:**

- The fitted images in the output files
- The fitted Zernike coefficients
- And ultimately the computed m1 and m2 commands to be sent, summarized below:

Image timestamp	Commands
T00%3A18%3A00.xx	m2commands: [759 -71 -43 65 28] m1commands: [42 1 20 18 0 0 0 0 0 0 0 0 0 0 0 0]
T00%3A19%3A20.xx	m2commands: [77 -32 -32 28 -9] m1commands: [28 -71 42 15 0 0 0 0 0 0 0 0 0 0 0 0]
T00%3A20%3A37.xx	m2commands: [6 -35 -9 -12 -5] m1commands: [9 -4 -43 -27 0 0 0 0 0 0 0 0 0 0 0 0]
T00%3A22%3A56.xx	m2commands: [33 -31 -162 -49 4] m1commands: [-85 -85 -155 123 0 0 0 0 0 0 0 0 0 0 0 0]
T00%3A26%3A48.xx	m2commands: [-188 139 6 -24 -43] m1commands: [116 69 101 73 0 0 0 0 0 0 0 0 0 0 0 0]
T00%3A28%3A15.xx	m2commands: [94 101 -7 -15 -34] m1commands: [58 -60 12 -102 0 0 0 0 0 0 0 0 0 0 0 0]