

Fully commented and exemplified Mathematica code for Data Reduction of Seeing data – on the wave crest (without lobe work)

[Comments in black; code in blue; output in red]

(* The first part of the data analysis will look at the flat - calibrated raw data coming from the CCD (input of text file in ReadList); the idea is to have a good 3D view of it, where the vertical axis are the ADUs of each pixel *)

(* 1st, we need to input the matrix (x, y) size of the CCD chip *)

(* Here, and elsewhere in the routine, relevant numbers will be printed out so that after saving the file these can be looked up later; we start doing so for the [x, y] CCD chip size *)

```
matx = Input["Please enter the x-direction number of pixels in the CCD chip:"]; maty = Input["Please enter the y-direction number of pixels in the CCD chip:"]; Print["CCD chip size (x,y): (", matx, ",", maty, ")"];
```

CCD chip size (x,y): (765,510)

(* Routine 1 - Initial look at the data *)

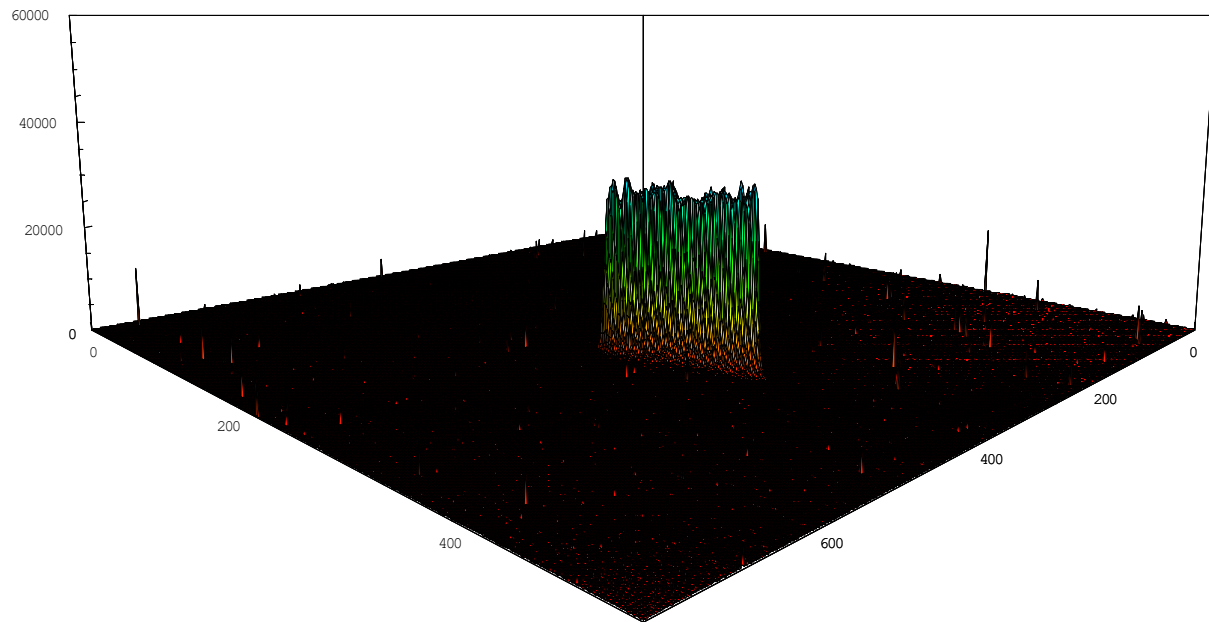
(* The 1st plot will give an idea of where the data lie in the (x, y, ADU) space; note that the initial ADU maximum is set at 60000, but this might be changed higher (if the data come out chopped) or lower (closer to the maximum ADU value); the maximum value of ADUs, that the routine also gives as output, helps in this matter; only change the viewpoint as a last resource *)

(* The 1st instruction is actually to ignore spell checking since it can be annoying for similarly-named variables *)

```
Off[General::spell1]
```

(* Obviously, change the SetDirectory according to the place where you have the CCD text file; and place its name on "ReadList" *)

```
SetDirectory["C:\Users\augus\Pedro\Pedro\Investigacao\Paper_ScienceAdvances_LOwCOstSEeing\Paper\Paper_Manuscript\Routines_and_Instructions"];  
a = ReadList["Calibrated_sample_data_040708_A2.txt", Table[Number, {matx}]]; ListPlot3D[a, ColorFunction -> Hue, PlotRange -> {{0, matx}, {0, maty}, {0, 60000}}, ViewPoint -> {1, 1, .2}]; m = Max[a]; Print["Peak value (ADU): ", m]
```

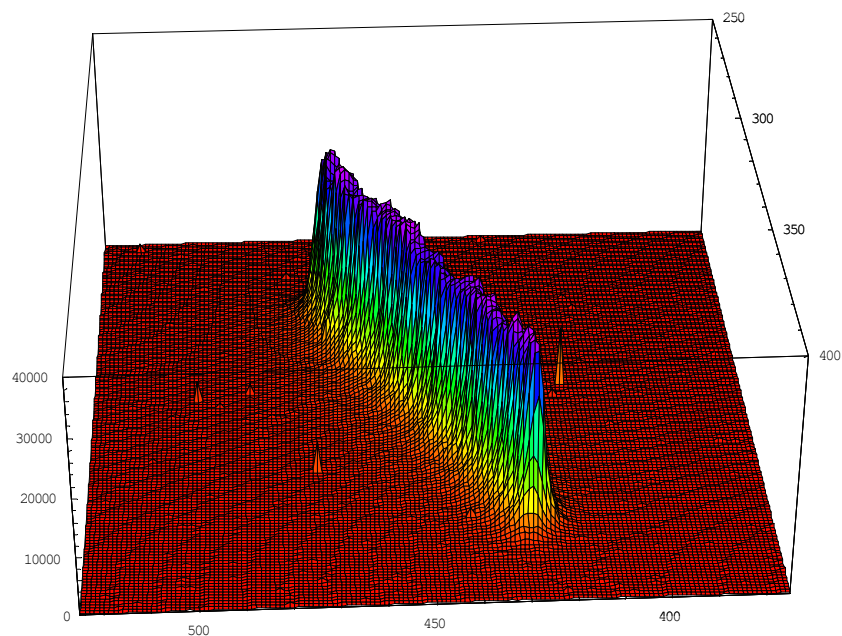


Peak value (ADU): 33159

(* Routine 2 - Interact with the data : final graph production and view *)

(* Given the peak value, the following routine will have to be adjusted until we have a "nice looking" 3D plot of the Polaris path; we usually limit the maximum to the next "ten of thousand" and guess a tighter SQUARE (x1, x2) - (y1, y2) matrix from the plot above, eventually enlarging it : trial an error; in the end, changing the viewpoint might be necessary too *)

`ListPlot3D[a, ColorFunction -> Hue, PlotRange -> {{375,525}, {250, 400}}, {0, 40000}], ViewPoint -> {.2, 4, 2.5}]`



-SurfaceGraphics-

(* Once you have settled on a fine looking plot, the final values must be inserted by running the routine that follows; the print out confirms them *)

```
nx1 = Input["x1?"]; nx2 = Input["x2?"]; ny1 = Input["y1?"]; ny2 = Input["y2?"]; Print["x1= ", nx1, ",  
x2= ", nx2, ", y1= ", ny1, ", y2= ", ny2];
```

x1= 375, x2= 525, y1= 250, y2= 400

(* The final (x1 - x2) - (y1 - y2) values are now inserted, and there we go! *)

(* Routine 3 (Main) - Organizing the data into 2x2 pixels, preparing vectors c and f for analysis; be patient, it might take a few minutes! *)

(* The following routine will identify the coordinates (x, y) of the maximum value in ADU (m); right now we also define, from this value, the maximum range for the contour plots - topgr *)

```
b = Transpose[a]; x = 1; y = 1; mx = 0; my = 0;  
While[y < maty + 1,  
  While[x < matx + 1, If[b[[x, y]] == m, mx = x; my = y; x = matx + 1; y = maty + 1, x = x + 1,  
    Print["erro"]]]; x = 1; y = y + 1];  
topgr = m + 5000;
```

(* Now, starting with the maximum value, we define the 1st 2x2 'group' which has the 1st maximum, 2nd maximum (close), 3rd maximum, etc. all together - other values further away may be higher but we use the peak as an 'anchor' for defining the "peak 2x2 group"; the 2x2 peak will be defined within the eight pixels that surround the peak *)

(* We first find the 2nd maximum - m2 *)

```
x = mx - 1; y = my - 1; i = 0; j = 0; u = 100;  
While[y < my + 2, While[x < mx + 2, If[b[[x, y]] > u, u = b[[x, y]]; i = x; j = y; x = x + 1]; y = y + 2;  
x = mx - 1; If[b[[mx - 1, my]] > u, u = b[[mx - 1, my]]; i = mx - 1; j = my]; If[b[[mx + 1, my]] > u,  
u = b[[mx + 1, my]]; i = mx + 1; j = my]; m2 = u; m2x = i; m2y = j;
```

(* Now, if the 2nd maximum is diagonal as regards the peak m, the 2x2 peak is immediately defined; we check this next ; note that we define the 2x2 peak as a new pixel in the 1st element of a new matrix d where the ADU is the average of the four values and the coordinates of the 2x2 pixel are the ones of the point at the intersection of the four pixels; similarly, another matrix e is built (1st element) containing the standard deviations of the calculus *)

(* Now inserted into the 3rd IF of the one above (option if test is false) we work the more difficult situation of the 2nd maximum not being in a diagonal; in this case, it is either left, up, right or down from the main peak; to decide, we will have to look for the 3rd peak in the other two pixels that will complete the 2x2; two hypothesis in each case*)

```

esq = 0; drt = 0; cim = 0; bxo = 0; dx = 0; dy = 0; dm = 0; es = 0; d = {}; e = {}; i = 0; j = 0; If[Abs[mx -
m2x] == 1, i = 1, i = 0]; If[Abs[my - m2y] == 1, j = 1, j = 0];
If[i + j == 2, If[mx > m2x && m2y > my, dx = mx - 0.5; dy = my + 0.5; dm = N[Mean[{m, m2, b[[mx - 1,
my]], b[[mx, my + 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx - 1, my]], b[[mx, my + 1]]]}; d =
Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}],
    If[m2x > mx && m2y > my, dx = mx + 0.5; dy = my + 0.5; dm = N[Mean[{m, m2, b[[mx + 1, my]],
b[[mx, my + 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx + 1, my]], b[[mx, my + 1]]]}; d =
Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}],
    If[m2x > mx && my > m2y, dx = mx + 0.5; dy = my - 0.5; dm = N[Mean[{m, m2, b[[mx + 1, my]],
b[[mx, my - 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx + 1, my]], b[[mx, my - 1]]]}; d =
Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}],
    If[mx > m2x && my > m2y, dx = mx - 0.5; dy = my - 0.5; dm = N[Mean[{m, m2, b[[mx - 1,
my]], b[[mx, my - 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx - 1, my]], b[[mx, my - 1]]]}; d =
Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}]]],
If[m2x == mx && m2y > my, esq = Max[b[[mx - 1, my + 1]], b[[mx - 1, my]]]; dir = Max[b[[mx + 1, my
+ 1]], b[[mx + 1, my]]];
    If[esq > dir, dx = mx - 0.5; dy = my + 0.5; dm = N[Mean[{m, m2, b[[mx - 1, my]], b[[mx - 1, my +
1]]]}; es = N[StandardDeviation[{m, m2, b[[mx - 1, my]], b[[mx - 1, my + 1]]]}; d = Append[d, {dx, dy,
dm}]; e = Append[e, {dx, dy, es}], dx = mx + 0.5; dy = my + 0.5; dm = N[Mean[{m, m2, b[[mx + 1,
my]], b[[mx + 1, my + 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx + 1, my]], b[[mx + 1, my + 1]]]}; d =
Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}]]];
If[m2x == mx && m2y < my, esq = Max[b[[mx - 1, my - 1]], b[[mx - 1, my]]]; dir = Max[b[[mx + 1, my
- 1]], b[[mx + 1, my]]];
    If[esq > dir, dx = mx - 0.5; dy = my - 0.5; dm = N[Mean[{m, m2, b[[mx - 1, my]], b[[mx - 1, my -
1]]]}; es = N[StandardDeviation[{m, m2, b[[mx - 1, my]], b[[mx - 1, my - 1]]]}; d = Append[d, {dx, dy,
dm}]; e = Append[e, {dx, dy, es}], dx = mx + 0.5; dy = my - 0.5; dm = N[Mean[{m, m2, b[[mx + 1, my]],
b[[mx + 1, my - 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx + 1, my]], b[[mx + 1, my - 1]]]}; d =
Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}]]];
If[m2y == my && m2x < mx, cim = Max[b[[mx - 1, my + 1]], b[[mx, my + 1]]]; bxo = Max[b[[mx - 1, my
- 1]], b[[mx, my - 1]]];
    If[cim > bxo, dx = mx - 0.5; dy = my + 0.5; dm = N[Mean[{m, m2, b[[mx - 1, my + 1]], b[[mx, my +
1]]]}; es = N[StandardDeviation[{m, m2, b[[mx - 1, my + 1]], b[[mx, my + 1]]]}; d = Append[d, {dx,
dy, dm}]; e = Append[e, {dx, dy, es}], dx = mx - 0.5; dy = my - 0.5; dm = N[Mean[{m, m2, b[[mx - 1,
my - 1]], b[[mx, my - 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx - 1, my - 1]], b[[mx, my - 1]]]};
d = Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}]]];
If[m2y == my && m2x > mx, cim = Max[b[[mx + 1, my + 1]], b[[mx, my + 1]]]; bxo = Max[b[[mx + 1, my
- 1]], b[[mx, my - 1]]];
    If[cim > bxo, dx = mx + 0.5; dy = my + 0.5; dm = N[Mean[{m, m2, b[[mx + 1, my + 1]], b[[mx, my +
1]]]}; es = N[StandardDeviation[{m, m2, b[[mx + 1, my + 1]], b[[mx, my + 1]]]}; d = Append[d, {dx,
dy, dm}]; e = Append[e, {dx, dy, es}], dx = mx + 0.5; dy = my - 0.5; dm = N[Mean[{m, m2, b[[mx + 1,
my - 1]], b[[mx, my - 1]]]}; es = N[StandardDeviation[{m, m2, b[[mx + 1, my - 1]], b[[mx, my - 1]]]};
d = Append[d, {dx, dy, dm}]; e = Append[e, {dx, dy, es}]]];

```

(* Now we organize all data into 2x2 pixels; special care has to be taken when reaching the corners and this is why the previous work was done : peak identification allows to decide where to chop the data in x and y in order to ensure that both axis have a number of pixels divisible by four; note that we initialize the matrices d and e so that they are ready to remap the CCD chip into 2x2 ; we also

create vector d1 (and its transpose d3) so that we can later plot the fit on top of the 2x2 data, as well as on top of the original 1x1 data*)

(* this routine takes a few minutes to run *)

```
mx = dx; my = dy; m = dm; ordd = 0; ul = 0; ur = 0; dl = 0; dr = 0; i = dx - 0.5; j = dy - 0.5; dx = 0; dy = 0; dm = 0; es = 0; xlo = 1; xhi = matx; ylo = 1; yhi = maty; d = {}; e = {}; d1 = {}; d2 = {}; If[EvenQ[IntegerPart[j]] == True, ylo = 2; yhi = maty - 1]; If[EvenQ[IntegerPart[i]] == False, xhi = matx - 1, xlo = 2]; y = ylo; v = yhi; x = xlo; l = xhi; While[y < v + 1, While[x < l + 1, dl = b[[x, y]]; dr = b[[x + 1, y]]; ul = b[[x, y + 1]]; ur = b[[x + 1, y + 1]]; dx = x + 0.5; dy = y + 0.5; dm = N[Mean[{dl, dr, ul, ur}]]; es = N[StandardDeviation[{dl, dr, ul, ur}]]; d = Append[d, {dx, dy, dm}]; d2 = Append[d2, dm]; d2 = Append[d2, dm]; If[dx == mx && dy == my, ordd = Length[d]]; e = Append[e, {dx, dy, es}]; x = x + 2]; x = xlo; d1 = Append[d1, d2]; d1 = Append[d1, d2]; d2 = {}; y = y + 2]; d3 = Transpose[d1];
```

(* Now, we build the main routine: starting from the peak and building up the path both ways, until reaching a value <10000 ADU; always picking the highest-valued pixel around*)

(* We first do it for the peak only; this will define the direction of the initial path (2nd maximum-m2); the second similar subroutine does the other bit (3rd maximum-m3) that will be used to complete the path later, in the opposite direction *)

```
c = {{mx, my, m}}; k = 0; l = 0; m2xyADU = 0; u = 10001; ordem = ordd - IntegerPart[(matx + 3)/2]; While[l < 2, While[k < 3, ordem = ordem + 1; If[d[[ordem, 3]] > 10000 && d[[ordem, 3]] > u, u = d[[ordem, 3]]; m2xyADU = ordem]; k = k + 1]; k = 0; ordem = ordem + matx - 4; l = l + 1]; ordem = ordd - 3; While[k < 2, ordem = ordem + 2; If[d[[ordem, 3]] > 10000 && d[[ordem, 3]] > u, u = d[[ordem, 3]]; m2xyADU = ordem]; k = k + 1]; m2 = u; m2x = d[[m2xyADU, 1]]; m2y = d[[m2xyADU, 2]]; If[m2 > 10000, c = Append[c, {m2x, m2y, m2}]];
```

```
k = 0; l = 0; m3xyADU = 0; u = 10001; ordem = ordd - IntegerPart[(matx + 3)/2]; f = {}; While[l < 2, While[k < 3, ordem = ordem + 1; If[d[[ordem, 3]] > 10000 && d[[ordem, 3]] > u && d[[ordem, 1]] != m2x && d[[ordem, 2]] != m2y, u = d[[ordem, 3]]; m3xyADU = ordem]; k = k + 1]; k = 0; ordem = ordem + matx - 4; l = l + 1]; ordem = ordd - 3; While[k < 2, ordem = ordem + 2; If[d[[ordem, 3]] > 10000 && d[[ordem, 3]] > u && d[[ordem, 1]] != m2x && d[[ordem, 2]] != m2y, u = d[[ordem, 3]]; m3xyADU = ordem]; k = k + 1]; m3 = u; m3x = d[[m3xyADU, 1]]; m3y = d[[m3xyADU, 2]]; If[m3 > 10000, f = Append[f, {m3x, m3y, m3}]];
```

(* We now have the main routine (currently limited to 100 cycles, otherwise it might enter an infinite cycle mode before reaching low pixels; if it reaches values < 10000 ADU before the 100 cycles it will conclude with error(s) ... [don't worry]); this will start from the 2nd maximum and explore only three pixels; if m - m2 is a diagonal, these are the "corner" arrow; otherwise they will be in a line; the routine stops when reaching a value lower than 10000 (st = 1); then we will do the same in a separate routine (for now) for the third maximum (m3) so that we explore the other

direction from the peak (m); a later graph inspection might justify running the routine in "chunks" (if there are some < 10000 ADU areas that split higher values ones) : what we call "lobe work" *)

```

st = 0; temporario = 0; ciclos1 = 0; m1xyADU = ordd; m2ADU = m2xyADU; val382 =
IntegerPart[(matx - 1)/2];
While[st == 0 && ciclos1 < 100, orddif = m2xyADU - ordd; tempor = m2xyADU; u = 0; chave =
Abs[val382 - Abs[orddif]]; ordem = 0; teste = 0;
  If[d[[ordd + 2*orddif, 3]] > 10000 && d[[ordd + 2*orddif, 3]] > u, u = d[[ordd + 2*orddif, 3]];
  ordem = ordd + 2*orddif];
  If [chave == 1, teste = orddif - val382;
    If[teste == 1 || teste == -matx + 2,
      If[d[[ordd + 2*orddif - 1, 3]] > 10000 && d[[ordd + 2*orddif - 1, 3]] > u, u = d[[ordd +
2*orddif - 1, 3]]; ordem = ordd + 2*orddif - 1];
      If[d[[ordd + orddif + 1, 3]] > 10000 && d[[ordd + orddif + 1, 3]] > u, u = d[[ordd + orddif +
1, 3]]; ordem = ordd + orddif + 1],
      If[d[[ordd + 2*orddif + 1, 3]] > 10000 && d[[ordd + 2*orddif + 1, 3]] > u, u = d[[ordd +
2*orddif + 1, 3]]; ordem = ordd + 2*orddif + 1];
      If[d[[ordd + orddif - 1, 3]] > 10000 && d[[ordd + orddif - 1, 3]] > u, u = d[[ordd + orddif - 1,
3]]; ordem = ordd + orddif - 1]],
    If[chave == 0,
      If[d[[ordd + 2*orddif + 1, 3]] > 10000 && d[[ordd + 2*orddif + 1, 3]] > u, u = d[[ordd +
2*orddif + 1, 3]]; ordem = ordd + 2*orddif + 1];
      If[d[[ordd + 2*orddif - 1, 3]] > 10000 && d[[ordd + 2*orddif - 1, 3]] > u, u = d[[ordd +
2*orddif - 1, 3]]; ordem = ordd + 2*orddif - 1],
      If[d[[tempor + (val382 + 1)*orddif, 3]] > 10000 && d[[tempor + (val382 + 1)*orddif, 3]] > u,
u = d[[tempor + (val382 + 1)*orddif, 3]]; ordem = tempor + (val382 + 1)*orddif];
      If[d[[tempor - (val382 - 1)*orddif, 3]] > 10000 && d[[tempor - (val382 - 1)*orddif, 3]] >
u, u = d[[tempor - (val382 - 1)*orddif, 3]]; ordem = tempor - (val382 - 1)*
orddif]]];
  m2 = u; m2x = d[[ordem, 1]]; m2y = d[[ordem, 2]]; m2xyADU = ordem; ordd = tempor; ciclos1 =
ciclos1 + 1;
  If[m2 > 10000, c = Append[c, {m2x, m2y, m2}], st = 1]];

```

```

Part::partd :
Part specification {{1.5, 2.5, 176.25}, {3.5, 2.5, 182.}, <<97025>>, {763.5, 508.5, 186.5}}[0, 1]
is longer than depth of object. More...

Part::partd :
Part specification {{1.5, 2.5, 176.25}, {3.5, 2.5, 182.}, <<97025>>, {763.5, 508.5, 186.5}}[0, 1]
is longer than depth of object. More...

Part::partd :
Part specification {{1.5, 2.5, 176.25}, {3.5, 2.5, 182.}, <<97025>>, {763.5, 508.5, 186.5}}[0, 2]
is longer than depth of object. More...

General::stop :
Further output of Part::partd will be suppressed during this calculation. More...

```

(* We now do the main routine for the 3rd maximum, exactly the same way ; however, we will build vector f so that we can later combine with vector c to make a single path for the Pole Star from our data (c + f) *)

(* once again, error(s) is ok *)

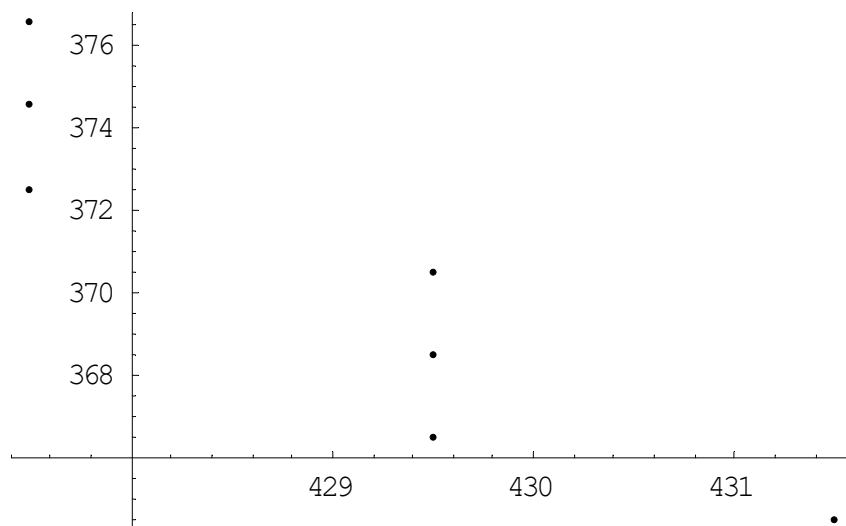
```
st = 0; ciclos2 = 0; orddif = 0; temporario = 0; ordd = m1xyADU; m3ADU = m3xyADU;
While[st == 0 && ciclos2 < 100, orddif = m3xyADU - ordd; temporario = m3xyADU; u =
0; chave = Abs[val382 - Abs[orddif]]; ordem = 0; teste = 0;
  If[d[[ordd + 2*orddif,3]] > 10000 && d[[ordd + 2*orddif, 3]] > u, u = d[[ordd + 2*orddif, 3]];
  ordem = ordd + 2*orddif];
  If[chave == 1, teste = orddif - val382;
    If[teste == 1 || teste == -matx + 2,
      If[d[[ordd + 2*orddif - 1,3]] > 10000 && d[[ordd + 2*orddif - 1, 3]] > u, u = d[[ordd +
2*orddif - 1, 3]]; ordem = ordd + 2*orddif - 1];
      If[d[[ordd + orddif + 1, 3]] > 10000 && d[[ordd + orddif + 1, 3]] > u, u = d[[ordd + orddif + 1,
3]]; ordem = ordd + orddif + 1],
      If[d[[ordd + 2*orddif + 1, 3]] > 10000 && d[[ordd + 2*orddif + 1, 3]] > u, u = d[[ordd +
2*orddif + 1, 3]]; ordem = ordd + 2*orddif + 1];
      If[d[[ordd + orddif - 1, 3]] > 10000 && d[[ordd + orddif - 1, 3]] > u, u = d[[ordd + orddif - 1,
3]]; ordem = ordd + orddif - 1]],
    If[chave == 0,
      If[d[[ordd + 2*orddif + 1, 3]] > 10000 && d[[ordd + 2*orddif + 1, 3]] > u, u = d[[ordd +
2*orddif + 1, 3]]; ordem = ordd + 2*orddif + 1];
      If[d[[ordd + 2*orddif - 1,3]] > 10000 && d[[ordd + 2*orddif - 1, 3]] > u, u = d[[ordd +
2*orddif - 1, 3]]; ordem = ordd + 2*orddif - 1],
      If[d[[temporario + (val382 + 1)*orddif,3]] > 10000 && d[[temporario + (val382 + 1)*orddif,
3]] > u, u = d[[temporario + (val382 + 1)*orddif, 3]]; ordem = temporario + (val382 + 1)*orddif];
      If[d[[temporario - (val382 - 1)*orddif, 3]] > 10000 && d[[temporario - (val382 - 1)*orddif,
3]] > u, u = d[[temporario - (val382 - 1)*orddif, 3]]; ordem = temporario - (val382 - 1)*orddif];
      m3 = u; m3x = d[[ordem, 1]]; m3y = d[[ordem, 2]]; m3xyADU = ordem; ordd = temporario; ciclos2
= ciclos2 + 1;
      If[m3 > 10000, f = Append[f, {m3x, m3y, m3}], st = 1];
```

```
Part::partd :
Part specification {{1.5, 2.5, 176.25}, {3.5, 2.5, 182.}, <<97025>>, {763.5, 508.5, 186.5}}[0, 1]
is longer than depth of object. More...

Part::partd :
Part specification {{1.5, 2.5, 176.25}, {3.5, 2.5, 182.}, <<97025>>, {763.5, 508.5, 186.5}}[0, 2]
is longer than depth of object. More...
```

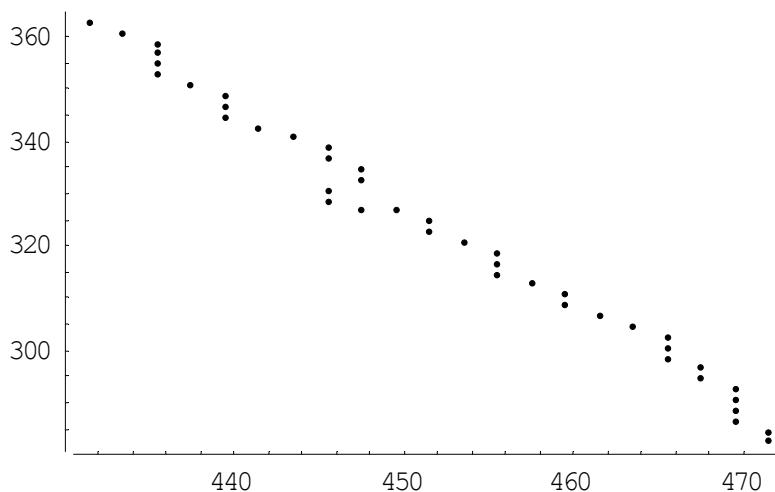
(* There is a final check to make, before establishing the final complete path : visually look (ideally, print) at both (x, y) graphs of c and f to make sure they will complete each other logically; we may decide new sections to remove; in that case, just go to routine 4 next and change the chopping pixel; for reference, the peak and 2nd peak have coordinates printed out here (1st graph), together with vector c in full; also the 3rd peak, in its own graph (and vector f in full) *)

```
l = 1; e = {}; n2 = Length[c];
While[l ≤ n2, e = Append[e, {c[[l, 1]], c[[l, 2]]}]; l = l + 1];
ListPlot[e, PlotRange -> All, PlotStyle -> PointSize[0.013]]; Print["Coordinates of the peak (x,y): ",
d[[m1xyADU, 1]], " ", d[[m1xyADU, 2]]]; Print["Coordinates of the 2nd maximum (x,y): ",
d[[m2ADU, 1]], " ", d[[m2ADU, 2]]]; Print[" Vector C: ", c]
```



Coordinates of the peak (x,y): 431.5 364.5
Coordinates of the 2nd maximum (x,y): 429.5 366.5
Vector C: {{431.5, 364.5, 32341.3}, {429.5, 366.5, 30779.3}, {429.5, 368.5, 31074.3}, {429.5, 370.5, 28874.8}, {427.5, 372.5, 31399.5}, {427.5, 374.5, 26287.}, {427.5, 376.5, 10671.5}}

```
l = 1; e = {}; n2 = Length[f];
While[l ≤ n2, e = Append[e, {f[[l, 1]], f[[l, 2]]}]; l = l + 1];
ListPlot[e, PlotRange -> All, PlotStyle -> PointSize[0.013]]; Print["Coordinates of the 3rd maximum (x,y): ", d[[m3ADU, 1]], " ", d[[m3ADU, 2]]]; Print["Vector F: ", f]
```



Coordinates of the 3rd maximum (x,y): 431.5 362.5
Vector F: {{431.5,362.5,30167.8},{433.5,360.5,29811.5},{435.5,358.5,30499.8},{435.5, 356.5,30828.5},{435.5,354.5,31493.},{435.5,352.5,31160.3},{437.5,350.5,30328.},{439.5,348.5,3045 8.5},{439.5,346.5,30562.8},{439.5,344.5,30127.},{441.5,342.5,30001.8},{443.5,340.5,29460.5},{445. 5,338.5,29457.8},{445.5,336.5,29570.8},{447.5,334.5,29750.3},{447.5,332.5,28347.5},{445.5,330.5, 29048.5},{445.5,328.5,15875.8},{447.5,326.5,17517.3},{449.5,326.5,30471.5},{451.5,324.5,31291.5} ,{451.5,322.5,32155.3},{453.5,320.5,30996.5},{455.5,318.5,31157.3},{455.5,316.5,30979.8},{455.5,3 14.5,30958.5},{457.5,312.5,31179.3},{459.5,310.5,30353.8},{459.5,308.5,30560.},{461.5,306.5,3025

```
5.8},{463.5,304.5,29504.5},{465.5,302.5,29635.8},{465.5,300.5,30302.3},{465.5,298.5,32019.8},{467
.5,296.5,31146.},{467.5,294.5,28406.3},{469.5,292.5,28110.},{469.5,290.5,30159.8},{469.5,288.5,30
841.5},{469.5,286.5,29801.3},{471.5,284.5,25518.},{471.5,282.5,18199.}}
```

(* Note that, before proceeding, clean versions of c and f must be created if the cycles reached the number 100: we have to remove all "repeat" sections ; the maximum point of interest (before repeats) MUST BE CHANGED IN THE ROUTINE FOR EACH CASE! *)

(* So, we first run the following routine, to check if that was the case; if one or two of the cycles reached 100 then the corresponding routine 4 that follows should be run to inspect the graph(s) of the relevant vector(s)*)

```
Print["Cycles of vector c, which includes the peak: ", ciclos1 + 1]; Print["Cycles of vector f, which
does not include the peak: ", ciclos2];
```

Cycles of vector c, which includes the peak: 7

Cycles of vector f, which does not include the peak: 42

(* Note that in the example used here, neither vector reached 100 cycles so we would jump straight to routine 5; however, routine 4 is also used to redefine one or two of vectors c/f; since, in this example, we want to do that, we will actually use it *)

(* In this example we will remove the last point of "f"+c ,{427.5,376.5,10671.5}, due to a 59% drop *)

(* Routine 4 (Cycles) - eventually redefining vector c and/or vector f *)

(* This routine might be used after graph inspection and/or if any of the cycles for vector c/f reached 100 *)

```
lma = Input["In the case of vector C cycles reaching 100 or in case you want to delete some point(s)
in vector C due to a larger than 33% drop/rise in vector h, please say in which point you want to
stop (number) or, if all is ok, write the number 0."]; If[lma ≠ 0, xcycl = {}; l = 1; While[l < lma + 1,
xcycl = Append[xcycl, c[[l]]]; l = l + 1]; Print["New Vector C: ", c = xcycl]
```

New Vector C: {{431.5, 364.5, 32341.3}, {429.5, 366.5, 30779.3}, {429.5, 368.5, 31074.3}, {429.5, 370.5, 28874.8}, {427.5, 372.5, 31399.5}, {427.5, 374.5, 26287.}}

```
mma = Input["In the case of vector F cycles reaching 100 or in case you want to delete some
point(s) in vector F due to a larger than 33% drop/rise in vector h, please say in which point you
want to stop (number) or, if all is ok, write the number 0."]; If[mma ≠ 0, ycycl = {}; m = 1; While[m <
mma + 1, ycycl = Append[ycycl, f[[m]]]; m = m + 1]; Print["New Vector F: ", f = ycycl]
```

(* Routine 5 (Path) *)

(* Now we join the two parts of the path; c includes the peak, f starts from the peak in the other direction but does not include it; we first create g that is f reversed *)

```
compf = Length[f]; k = 0; g = {}; i = 0;
While[k < compf, i = f[[compf - k]]; g = Append[g, i]; i = 0; k = k + 1];
```

(* We now create h, that combines c and f reversed (g), so that the full path is consistent and in one direction ; it is important to show vector h explicitly (for eventual main + lobe combinations without having to run the full Mathematica routine again; Max and Min are also presented); note that f might sometimes be ignored (or c) in which case we just do h = c (or h = f) in what follows *)

```
h = {}; h = Join[g, c]; Print["Vector h: ", h];
```

```
Vector h: {{471.5, 282.5, 18199.}, {471.5, 284.5, 25518.}, {469.5, 286.5, 29801.3}, {469.5, 288.5, 30841.5}, {469.5, 290.5, 30159.8}, {469.5, 292.5, 28110.}, {467.5, 294.5, 28406.3}, {467.5, 296.5, 31146.}, {465.5, 298.5, 32019.8}, {465.5, 300.5, 30302.3}, {465.5, 302.5, 29635.8}, {463.5, 304.5, 29504.5}, {461.5, 306.5, 30255.8}, {459.5, 308.5, 30560.}, {459.5, 310.5, 30353.8}, {457.5, 312.5, 31179.3}, {455.5, 314.5, 30958.5}, {455.5, 316.5, 30979.8}, {455.5, 318.5, 31157.3}, {453.5, 320.5, 30996.5}, {451.5, 322.5, 32155.3}, {451.5, 324.5, 31291.5}, {449.5, 326.5, 30471.5}, {447.5, 326.5, 17517.3}, {445.5, 328.5, 15875.8}, {445.5, 330.5, 29048.5}, {447.5, 332.5, 28347.5}, {447.5, 334.5, 29750.3}, {445.5, 336.5, 29570.8}, {445.5, 338.5, 29457.8}, {443.5, 340.5, 29460.5}, {441.5, 342.5, 30001.8}, {439.5, 344.5, 30127.}, {439.5, 346.5, 30562.8}, {439.5, 348.5, 30458.5}, {437.5, 350.5, 30328.}, {435.5, 352.5, 31160.3}, {435.5, 354.5, 31493.}, {435.5, 356.5, 30828.5}, {435.5, 358.5, 30499.8}, {433.5, 360.5, 29811.5}, {431.5, 362.5, 30167.8}, {431.5, 364.5, 32341.3}, {429.5, 366.5, 30779.3}, {429.5, 368.5, 31074.3}, {429.5, 370.5, 28874.8}, {427.5, 372.5, 31399.5}, {427.5, 374.5, 26287.}}
```

(* now, we plot the profile of the path, 4 - pixel per 4 - pixel; we create vector h3 which will be very relevant for curve fitting; note that the typical profile should have a \ or / or /\ shape, never \/; in this latter case recreate h by using the reverse order in Join above : Join[c, g] *)

(* In addition, carefully check the profile, since it must follow the path on the plot; when the path is too vertical, this is easily missed; so, we must sort the vector properly: BEFORE the full line "While...e=Append...l=1;" put : "e=h[[Ordering[h[[All,2]]]]];h=e;e={};" *)

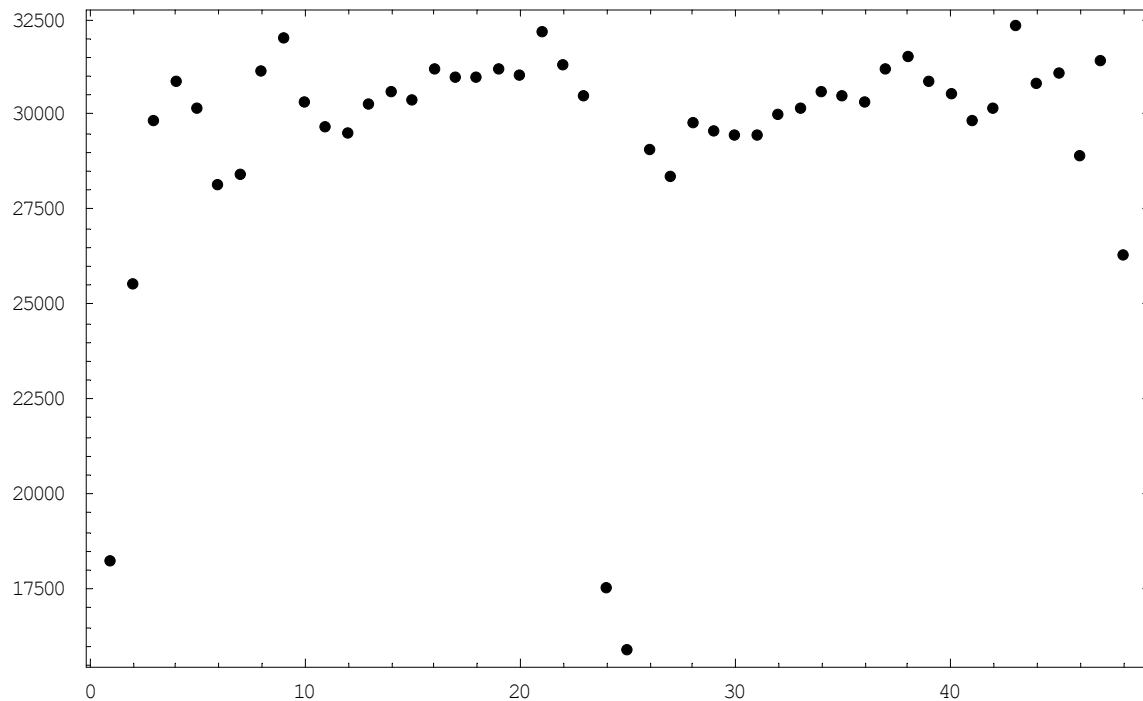
(* Now is also (yet another) moment to remove any extremity points that drop/rise more than 33% in relation to the previous/next point (use routine 4 above for it, as we have done in this example) *)

```
l = 1; h3 = {}; n = Length[h]; Print["Number of points in vector h: ", n]; e = {}; n2 = Length[h]; While[l ≤ n2, e = Append[e, {h[[l, 1]], h[[l, 2]]}]; l = l + 1]; l = 1; While[l ≤ n, h3 = Append[h3, h[[l, 3]]]; l = l + 1]; Print["Maximum 4-pix of vector h: ", Max[h3]]; Print["Minimum 4-pix of vector h: ", Min[h3]]; ListPlot[h3, PlotRange -> All, PlotStyle -> PointSize[0.013], Frame -> True, Axes -> False]
```

Number of points in vector h: 48

Maximum 4-pix of vector h: 32341.3

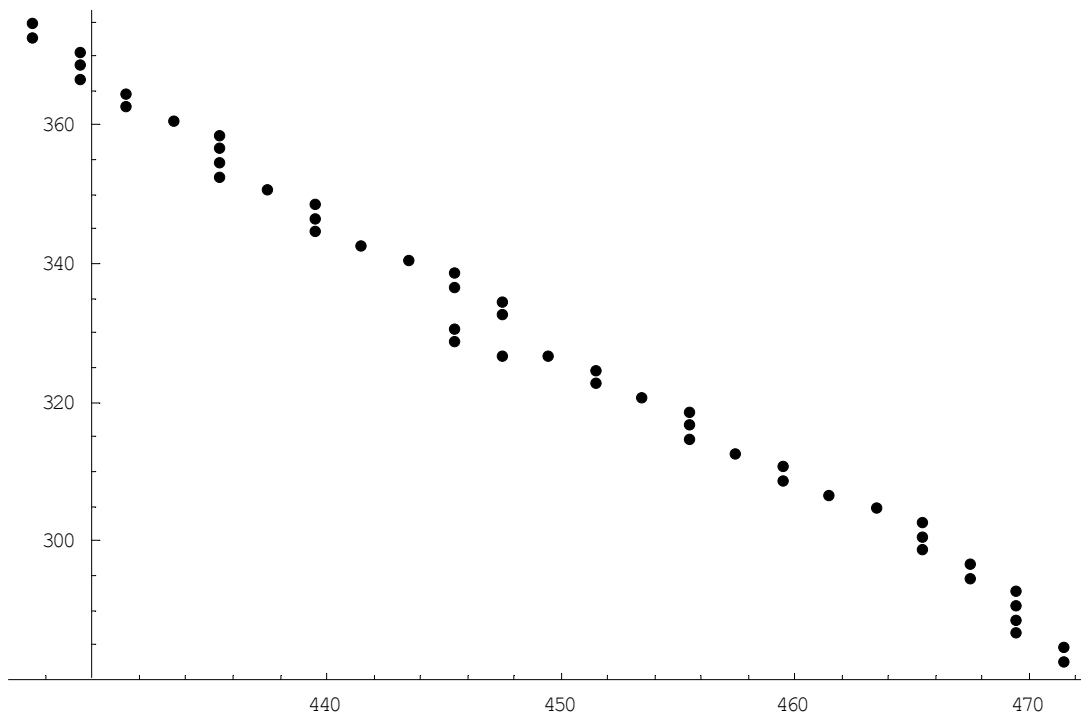
Minimum 4-pix of vector h: 15875.8



-Graphics-

(* We next plot the Pole Star path given by $h\{x, y\}$ (x is each first coordinate $h[i, 1]$ and y each second coordinate $h[i, 2]$); this will later be placed on top of the 3D graph (viewed as a 2D contour plot; presented after); all this must be done BEFORE curve fitting *)

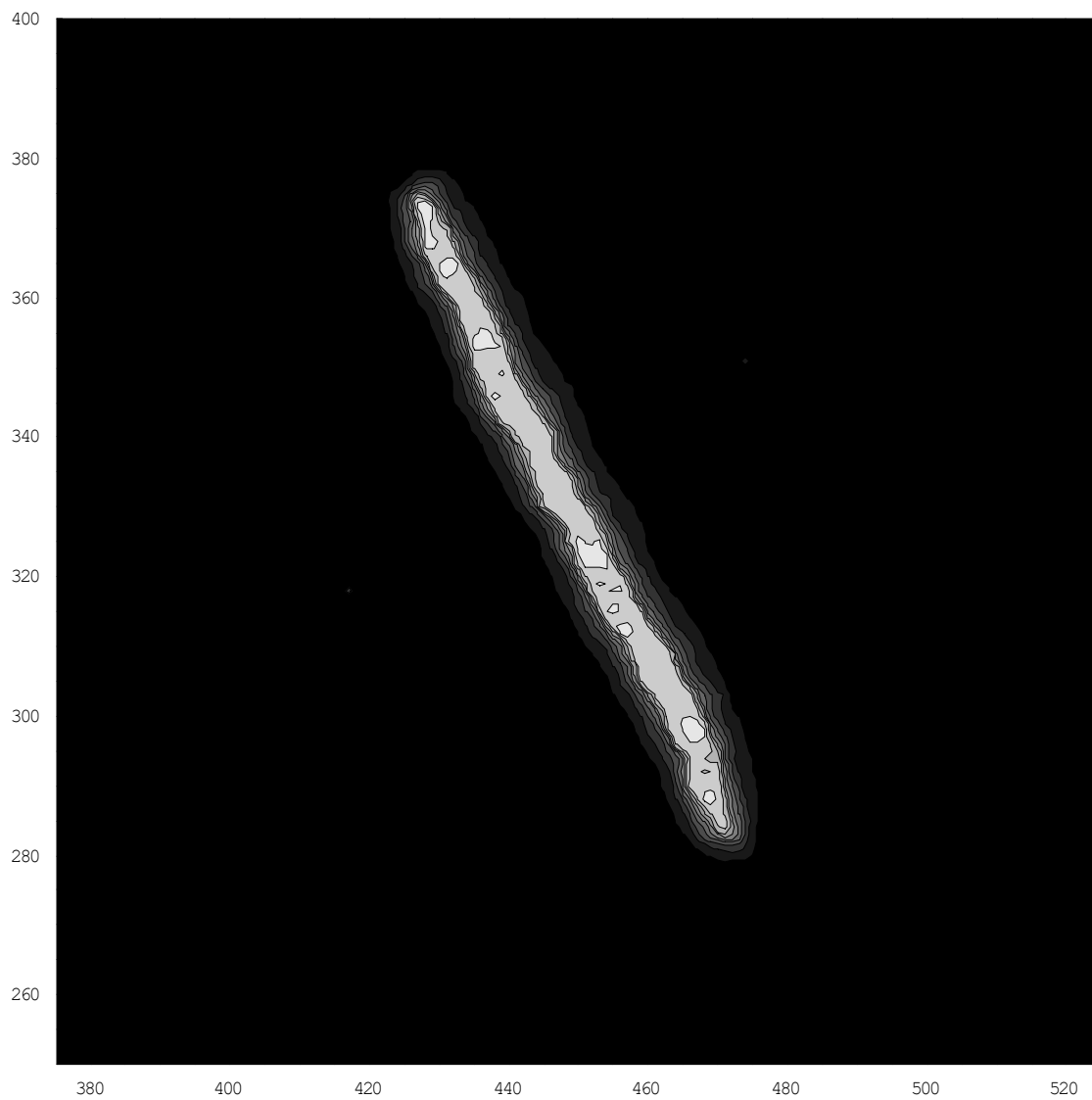
`e1 = ListPlot[e, PlotRange -> All, PlotStyle -> PointSize[0.013]]`



-Graphics-

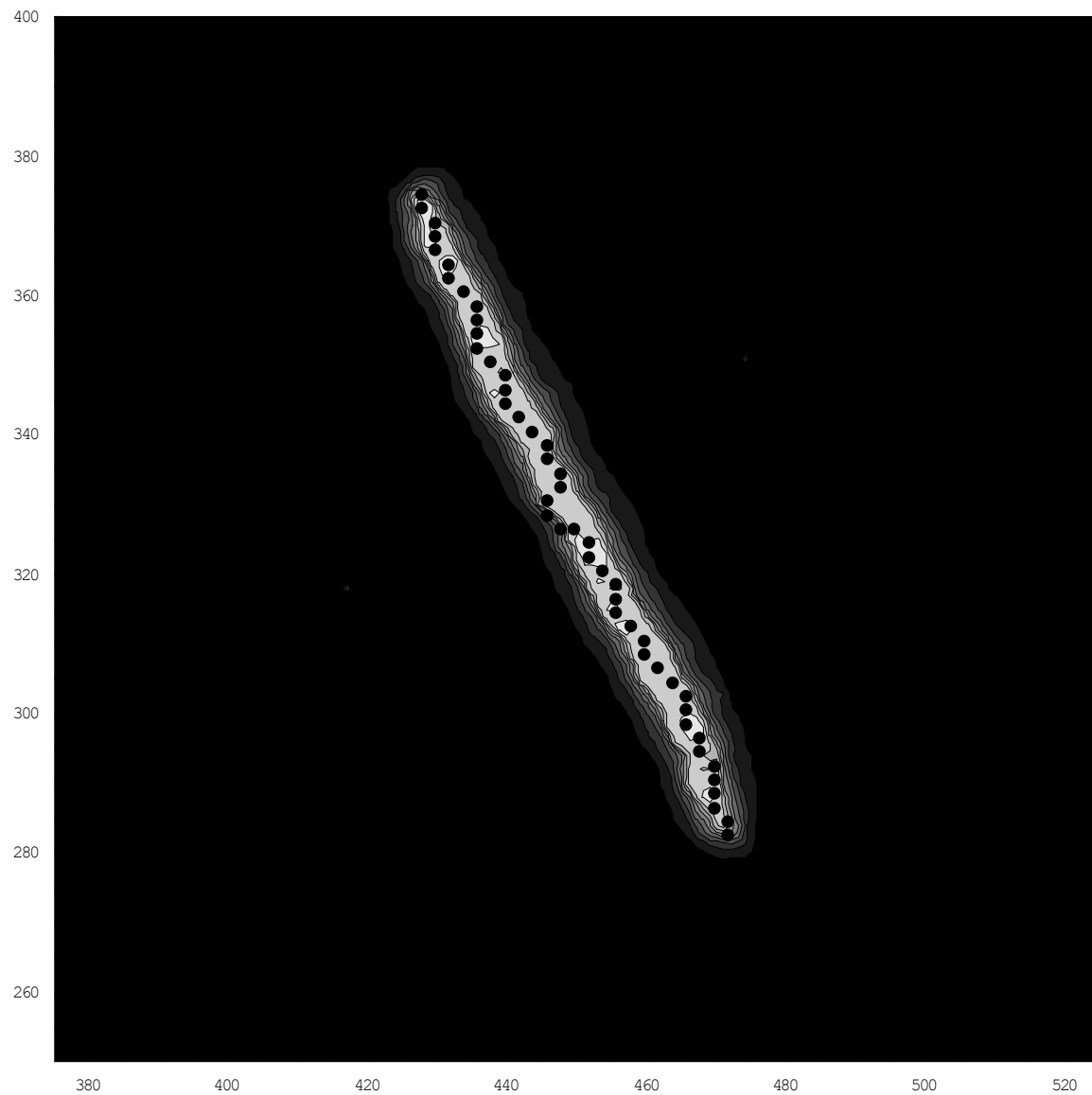
(* First the 1x1 superposition *)

```
e2 = ContourGraphics[a, PlotRange -> {{nx1,nx2}, {ny1, ny2}, {0, topgr}}]; Show[e2]
```



-ContourGraphics-

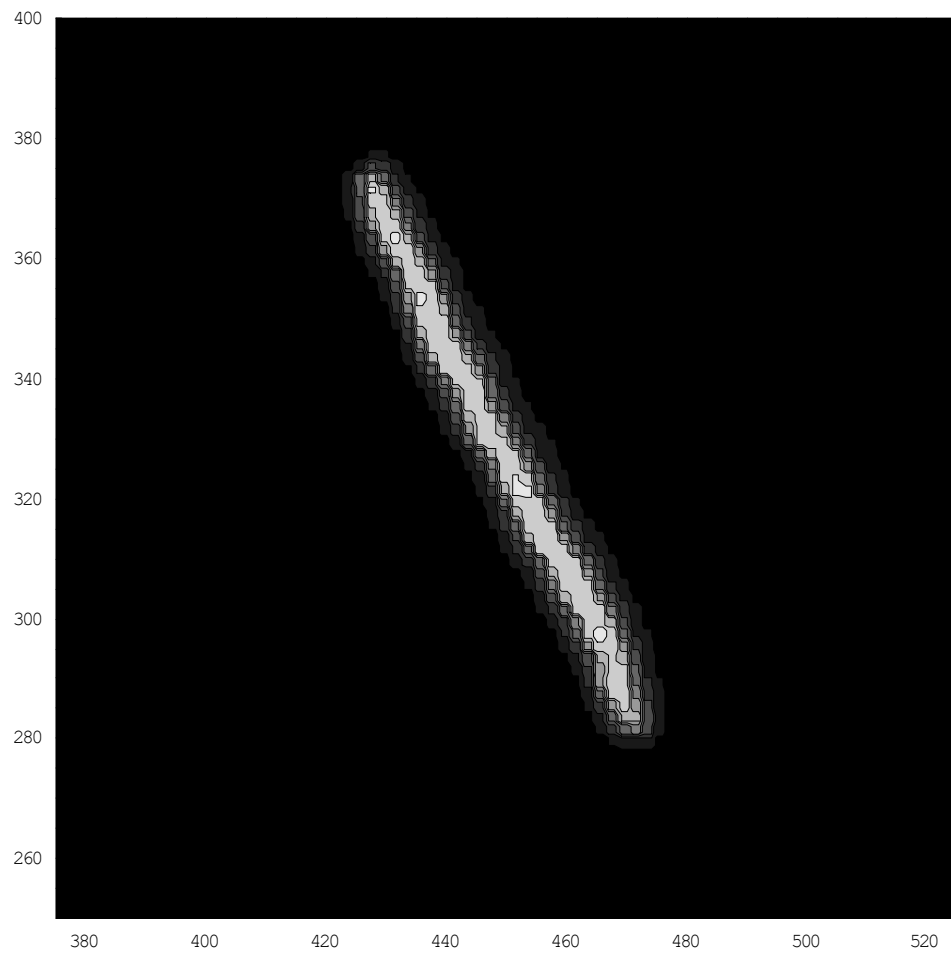
```
Show[e2, e1]
```



-Graphics-

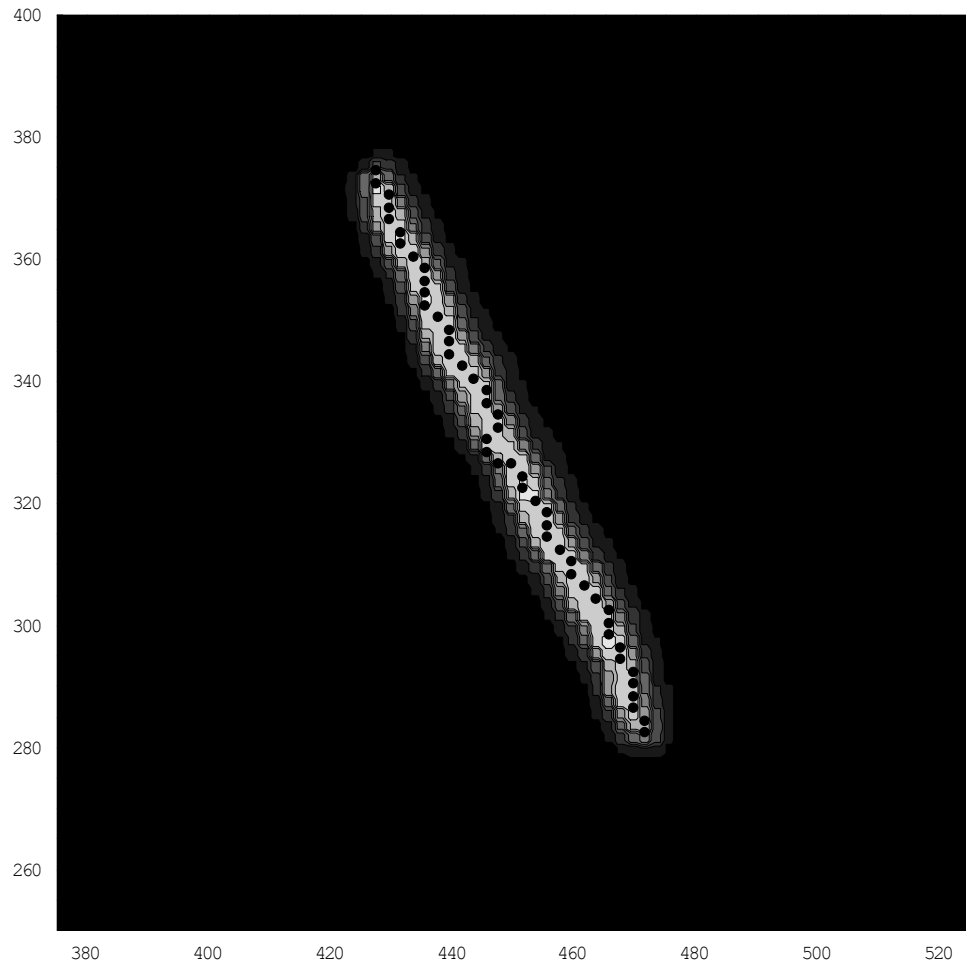
(* Now, the 2x2 superposition *)

```
q2=ContourGraphics[d1,PlotRange->{{nx1,nx2},{ny1,ny2},{0,topgr}}];Show[q2]
```



■ContourGraphics■

Show[q2, e1]



-Graphics-

(* Note that the seeing determination will take place only if there is no "LOBE work" to be made; if this is not the case, jump to the Special Lobe Section; otherwise, carry on with Routine 6 *)

(* Now begins the SEEING DETERMINATION section *)

(* Routine 6 (seeing) *)

(* Routine 6.1 (Circle Fitting) *)

(* The 1st attempt (default) will be with Circle Fitting *)

(* The first step is to use information we know: the radius of the circle; this is the exact distance of Polaris to the NCP, coming from the exact declination when we observed Polaris. Using the tables in cadastral.com we have for 1st Feb 2008 a declination of 89°18'24.2" and for 5th Mar 2009 89°18'37.2". The difference of 13" is irrelevant, giving an error of 0.5% in the radius, so we pick the mean distance to the NCP (2489.3") which, divided by the pixel size of 0.77" gives 3233 pixels (rounding to units). *)

(* Recalling that vector e contains the data in the format we need, (x,y) coordinates of the Polaris path in vector h, we then fit these data as follows *)

(* It is important to modify the values in UnitStep, since these focus the fit in the arc section where the data points lie, which is a very small section of the full circle; we use the (x1,x2), (y1,y2) limits used for the graphs above; note that there will actually be two fits (one with the '+' sign; the other with the '-' sign); sometimes one fails completely but, more typically, both are usually similar; the graphs should be analysed, to make sure the default '+' is always the best fit *)

(* It is very important to Sort e, which we do first; note that if the path is too vertical, Sort may not work properly; in that case, replace "Sort[e]" by "e[[Ordering[e[[All,2]]]]]" *)

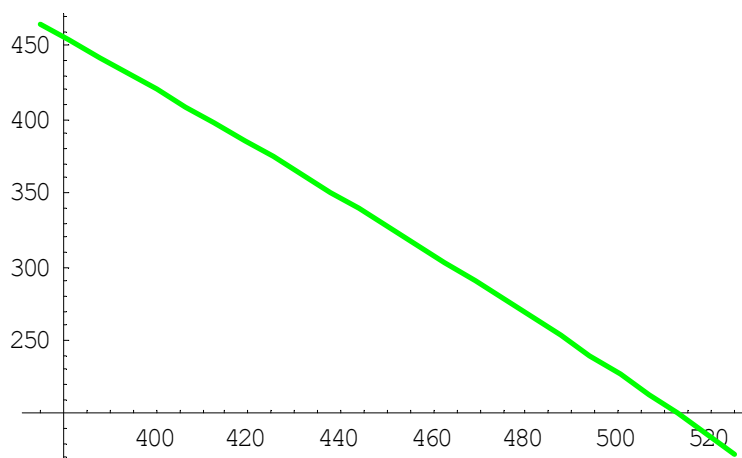
```
ess = {}; ess = Sort[e]; e = ess;
```

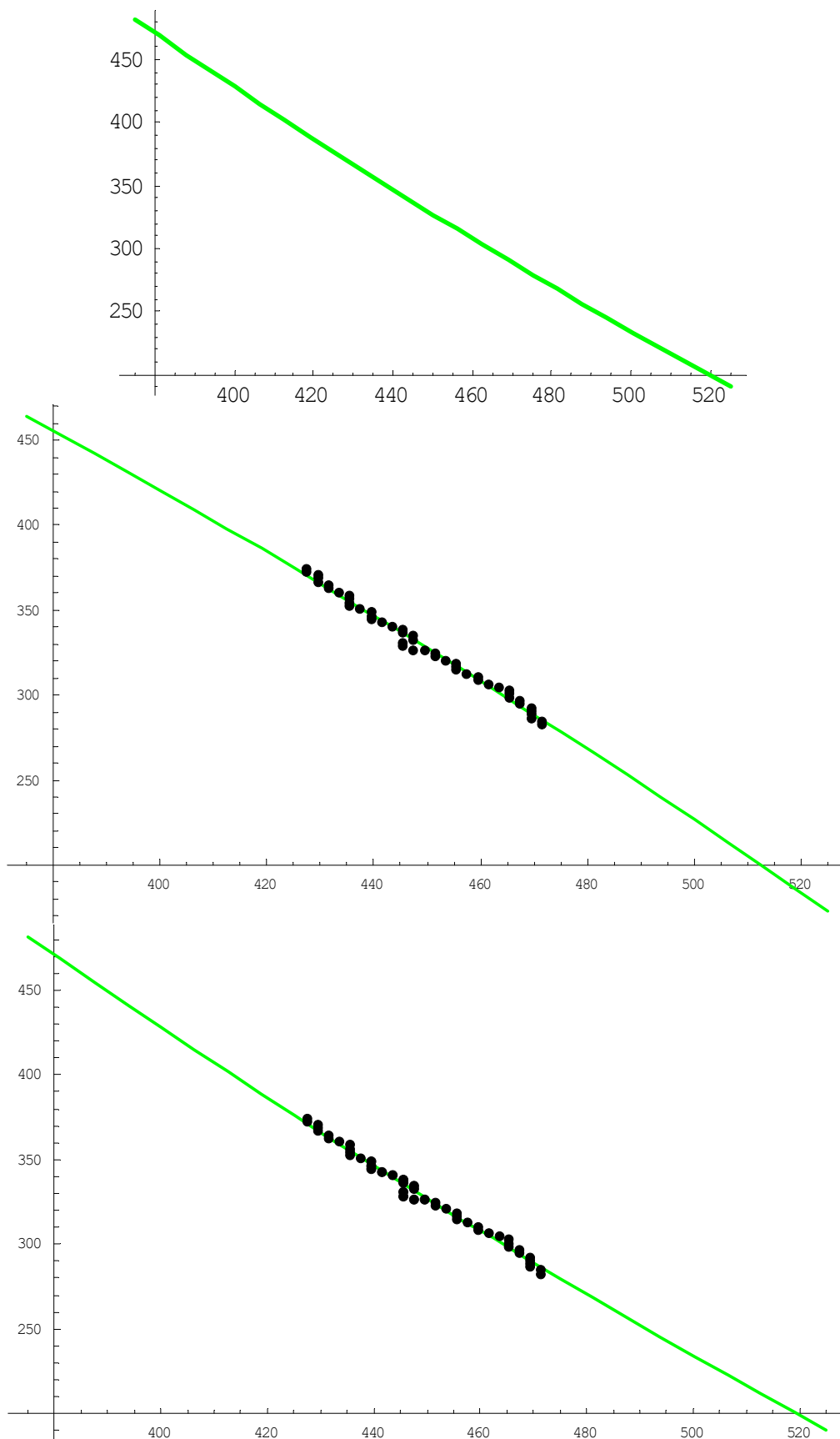
```
circfit1=FindFit[e,atr+Sqrt[3233^2-(UnitStep[r-nx1]*UnitStep[nx2-r]*r-btr)^2],{atr,btr},r,
MaxIterations->50000,Method->LevenbergMarquardt]; atrstr=ToString[circfit1[[1]]];
btrstr=ToString[circfit1[[2]]];latrstr=StringLength[atrstr];lbtrstr=StringLength[btrstr];
atr2=ToExpression[StringTake[atrstr,{8,latrstr}]];btr2=ToExpression[StringTake[btrstr,{8,lbtrstr}]];
```

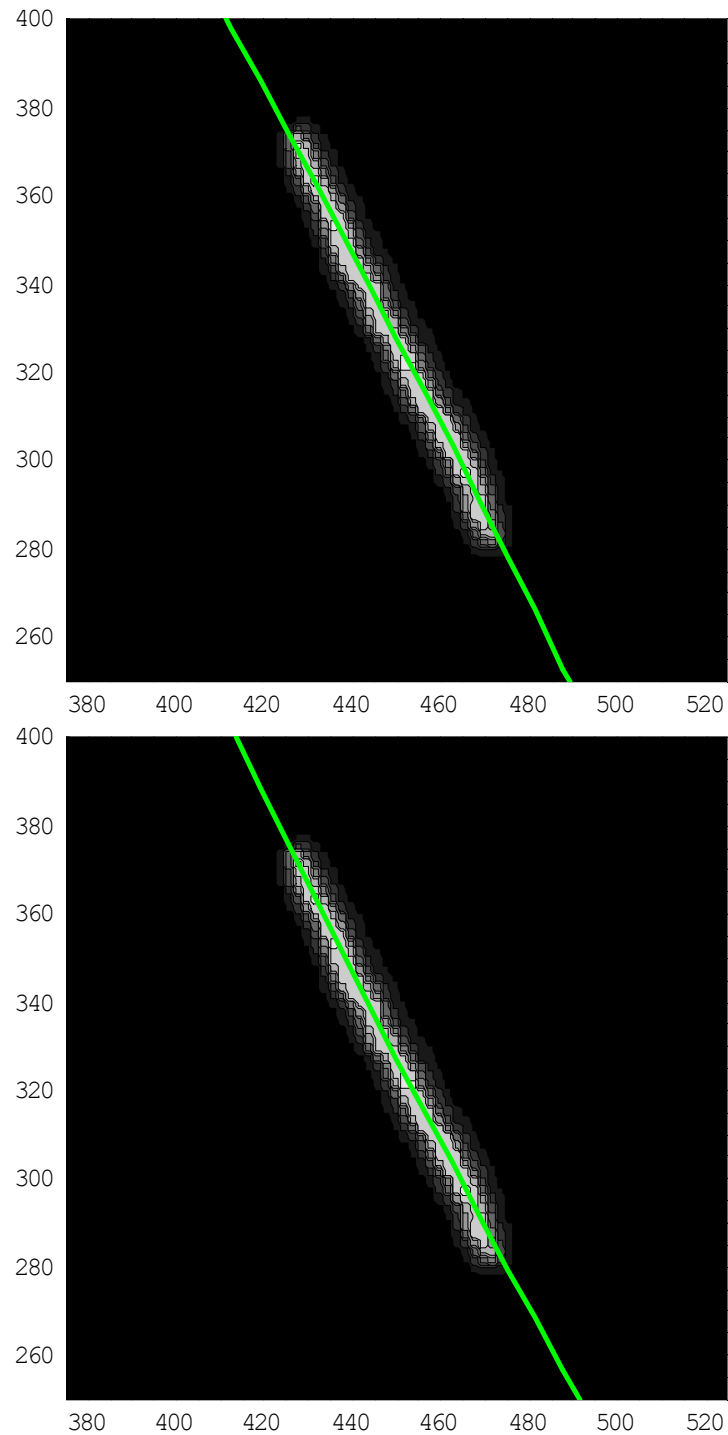
```
circfit2=FindFit[e,atm-Sqrt[3233^2-(UnitStep[r-nx1]*UnitStep[nx2-r]*r-btm)^2],{atm,btm},r,
MaxIterations->50000,Method->LevenbergMarquardt]; atmstr=ToString[circfit2[[1]]];
btmstr=ToString[circfit2[[2]]];latmstr=StringLength[atmstr];lbtmstr=StringLength[btmstr];
atm2=ToExpression[StringTake[atmstr,{8,latmstr}]];btm2=ToExpression[StringTake[btmstr,{8,lbtmstr}]];
```

(* We now build each graph fit *)

```
e10=Plot[atr2+Sqrt[3233^2-(r-btr2)^2],{r,nx1,nx2}, PlotStyle->{RGBColor[0,1,0],
AbsoluteThickness[2]}];
e20=Plot[atm2-Sqrt[3233^2-(r-btm2)^2],{r,nx1,nx2},
PlotStyle->{RGBColor[0,1,0],AbsoluteThickness[2]}];Show[e10,e1];Show[e20,e1];
Show[q2,e10];Show[q2,e20];
```







(* Finally, we get the seeing calculations for the circle fit; we pick the best from the two above *)

ans=Input["After careful eye inspection of both plots, in points and in greyscale, including curvature analysis, which one is better? If the 1st one (or the two cannot be distinguished), please place '1' below (only the number). If the 2nd one, please place '2'. "];

If[ans==2,at=atm2;bt=btm2;sinal=-1,at=atr2;bt=btr2;sinal=1];

(* We use the distance formula by constructing vector distancirc *)

```
l=1;n2=Length[h];distancirc={};somasqr=0;
While[l≤n2,sdf=Minimize[Sqrt[(t-h[[l,1]])^2+(at+sinal*Sqrt[Abs[3233^2-(t-bt)^2]]-
h[[l,2]])^2],t];distancirc=Append[distancirc,sdf[[1]]];
somasqr=somasqr+sdf[[1]]^2;l=l+1];seeingPh=Median[distancirc];seeingVis=2*seeingPh;
```

(* As you can see above, the routine just calculated the photographic and visual seeing values from the mean; however, we will print out (and consider) only the photographic R.M.S. value, which we calculate (and print out) next *)

```
seeingPh2=Sqrt[somasqr/n2]; seeingVis2=2*seeingPh2;sddev=seeingVis2*0.77/206265;
Print["CCD Seeing (R.M.S.) - CIRCLE FIT in arcseconds = ",seeingPh2*0.77]
```

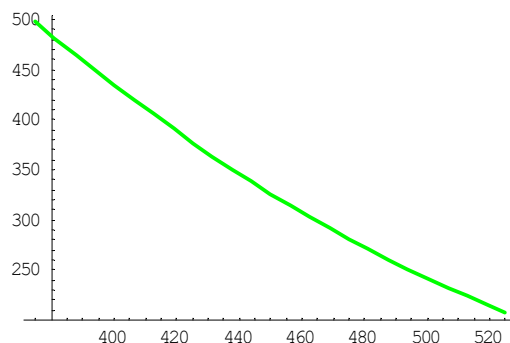
CCD Seeing (R.M.S.) - CIRCLE FIT in arcseconds = 0.934033

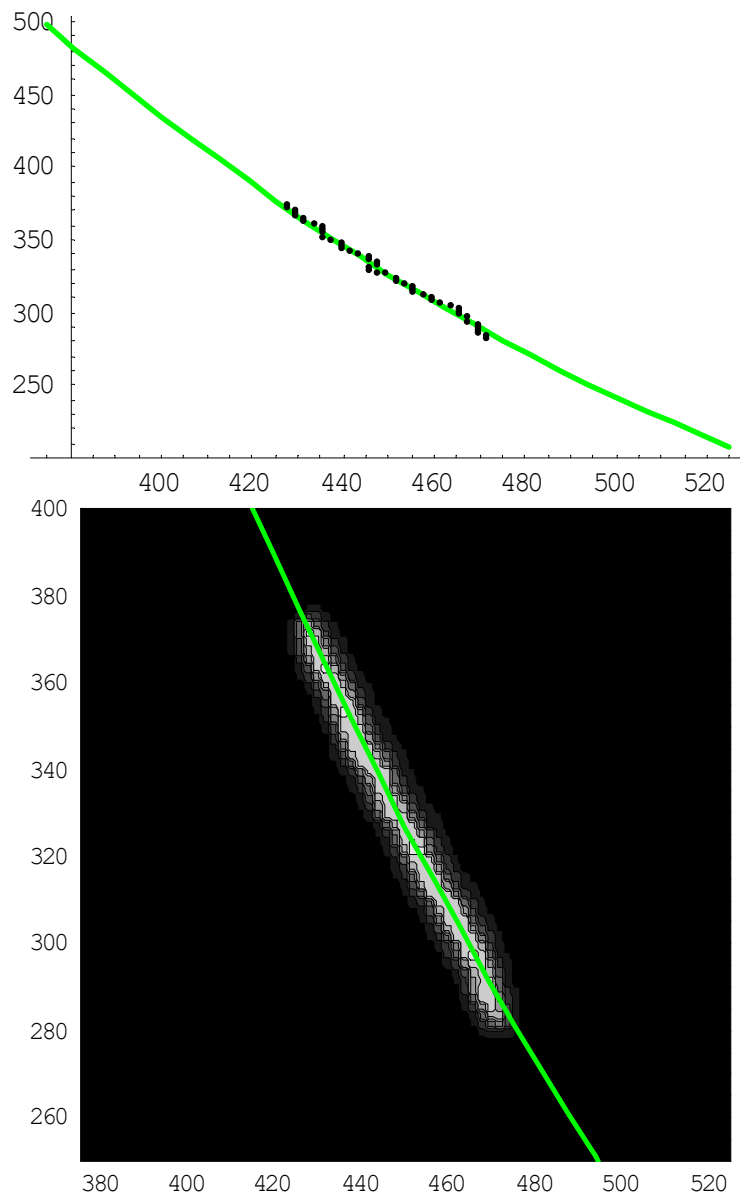
(* Routine 6.2 (Parabole Fitting) *)

(* For comparison, we have actually done a Parabole Fit too (when possible), following Moroder and Righini 1973; of course, such a fit is the only one available when Circle Fitting fails *)

(* The parabole fit is simple but not too physical; and, sometimes, the graphs are just stupid since we know Polaris is not following the full path that we see... It is, thus, a good idea to always get the alternative and more physical fit of a circle, since this is the actual motion of Polaris *)

```
parabfit1 = FindFit[e,a3 + a2 xp + a1 xp^2, {a3, a2, a1}, xp]; a3str = ToString[parabfit1[[1]]]; a2str =
ToString[parabfit1[[2]]]; a1str = ToString[parabfit1[[3]]]; la3str = StringLength[a3str]; la2str =
StringLength[a2str]; la1str = StringLength[a1str]; aa3 = ToExpression[StringTake[a3str, {7, la3str}]];
aa2 = ToExpression[StringTake[a2str, {7, la2str}]]; aa1 = ToExpression[StringTake[a1str, {7, la1str}]];
e3 = Plot[aa3 + aa2 xrt + aa1 xrt^2, {xrt, nx1, nx2}, PlotStyle -> {RGBColor[0, 1, 0],
AbsoluteThickness[2]}; Show[e3, e1]; Show[q2, e3]; l = 1; n2 = Length[e]; distancia = {};
somasqr = 0;
While[l ≤ n2, sdf = Minimize[Sqrt[(tpy - e[[l, 1]])^2 + (aa3 + aa2 tpy + aa1 tpy^2 - e[[l, 2]])^2], tpy];
distancia = Append[distancia, sdf[[1]]]; somasqr = somasqr + sdf[[1]]^2; l = l + 1]; seeingPh =
Median[distancia]; seeingVis = 2*seeingPh; seeingPh2 = Sqrt[somasqr/n2]; seeingVis2 =
2*seeingPh2; sddev = seeingVis2*0.77/206265; Print["CCD Seeing (R.M.S.) - PARABOLE FIT in
arcseconds = ", seeingPh2*0.77 ];
```





CCD Seeing (R.M.S.) - PARABOLE FIT in arcseconds = 0.874553

(* The seeing just calculated is the transversal one; Moroder & Righini 1973 use a different method to deduce the total seeing; however, they used photography (so, only statistically could they evaluate the deviations from the parabole), we use a CCD, much more precise and we could calculate the deviations exactly; also, we will not extrapolate to large telescopes as they did (our value is conservative anyway, this way - it would only get even smaller...) *)

(* Still, we can think a bit on the longitudinal seeing component; this is, no doubt, embedded in the brightness variations along the track; we are undersampling the pole star "Airy Disk" so we will have this in mind: some (unknown) part of the brightness variations is likely due to intrapixel variations; however, we would hope, in general, some relation between bad seeing and brightness variations (which would increase); for all these, at this stage, we will do these calculations and build a new vector fluctuations with all brightness variations, point after point; note that there is some

"attenuation" factor to bear in mind since horizontal or vertical motions along the chip will include more of the Airy disk in the 4-pixel set than 45 degree movements *)

(* We then start to determine the angle between the first and last point of h and the consequent reduction factor called fatorredutor; we will multiply this value by the final percentage calculated; we will compare each of the ADU values in the h vector extremes (the lowest ones) with the peak; the compensation factor will allow datasets to be comparable in the overall changes (percentages); this is also an issue for point-to-point fluctuations as we go along the vector h, although in this case only one angle out of horizontal/vertical is possible: 45 degrees *)

```
u1 = h[[1, 1]]; t1 = h[[1, 2]]; w1 = h[[n2, 1]]; v1 = h[[n2, 2]]; angyy = Abs[ArcTan[Abs[u1 - w1]/Abs[t1 - v1]]]; If[angyy > .78539816344, angyy = Pi/2 - angyy]; fatorredutor = Sqrt[1 + (Tan[angyy])^2];
```

(* Now we calculate the overall fluctuation, calibrated by the fatorredutor; if horizontal or vertical fatorredutor = 1; note that the typical path described by the vector h is from a low value to a maximum value to a low value again; even if this is not the case, we always use the extremes for the calculation; recall that the peak value (and its coordinates) are the first element of vector c *)

```
fluct1=(c[[1,3]]-h[[1,3]])*fatorredutor*100/c[[1,3]];
fluct2=(c[[1,3]]-h[[n2,3]])*fatorredutor*100/c[[1,3]]; Print["Peak-to-1st-path-point fluctuation:",fluct1, "%"]; Print["Peak-to-last-path-point fluctuation: ",fluct2, "%"];
```

Peak-to-1st-path-point fluctuation: 48.472%
Peak-to-last-path-point fluctuation: 20.7507%

(* Now, point - to - point fluctuations across h; we build vector flutuacoes; for simplicity we use the same angle formula as above, otherwise we would have to have a set of IFs; the final value will be a median of all values, just like we did for the seeing (transversal) *)

```
l=1;n2=Length[h];flutuacoes={};fluct=0;
While[l<n2,u1=h[[l,1]];t1=h[[l,2]];w1=h[[l+1,1]];v1=h[[l+1,2]];
  If[t1-v1≠0,angyy=Abs[ArcTan[Abs[u1-w1]/Abs[t1-v1]]];
  If[angyy>.78539816344,angyy=Pi/2-angyy]; fatorredutor=Sqrt[1+(Tan[angyy])^2],
  fatorredutor=1]; fluct=Abs[(h[[l,3]]-h[[l+1,3]])*fatorredutor]*100/h[[l,3]];
  flutuacoes=Append[flutuacoes,fluct];l=l+1]; seeinglong=Median[flutuacoes]; Print["Longitudinal 'seeing' (percentage of brightness variations along path): ",seeinglong, "%"]
```

Longitudinal 'seeing' (percentage of brightness variations along path): 2.68619%

(* Routine 6.3 ("Horizontal" Fitting) *)

(* The 3rd attempt, when Circle & Parabole fitting both fail, is the "Horizontal Method" *)

(* In the rare situation that the values of seeing are ridiculous (usually because the fitted arc of circle is too vertical) then run the following routine that calculates the horizontal distance of each point of vector h to the arc of circle *)

```

l = 1; n2 = Length[h]; distancirc = {}; xarcocirculo = {}; valor = 0; sdf = 0; somasqr = 0; ArcoCirculo = at
+ sinal*Sqrt[3233^2 - (t - bt)^2];
While[l ≤ n2, xarcocirculo = Solve[ArcoCirculo == h[[l, 2]], t]; valor = t /. xarcocirculo[[1]]; sdf =
Abs[valor - h[[l, 1]]]; distancirc = Append[distancirc, sdf]; somasqr = somasqr + sdf^2; l = l + 1;
xarcocirculo = {}; sdf = 0; valor = 0]; seeingPh = Median[distancirc]; seeingVis = 2*seeingPh;
seeingPh2 = Sqrt[somasqr/n2]; seeingVis2 = 2*seeingPh2; sddev = seeingVis2*0.77/206265;
Print["CCD Seeing (R.M.S.) - HORIZONTAL FIT in arcseconds = ", seeingPh2*0.77 ];

```