

Fully commented and exemplified Mathematica code for Data Reduction of Seeing data – FWHM

[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 *)

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

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

Off[General::spell1];

(* 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 *)

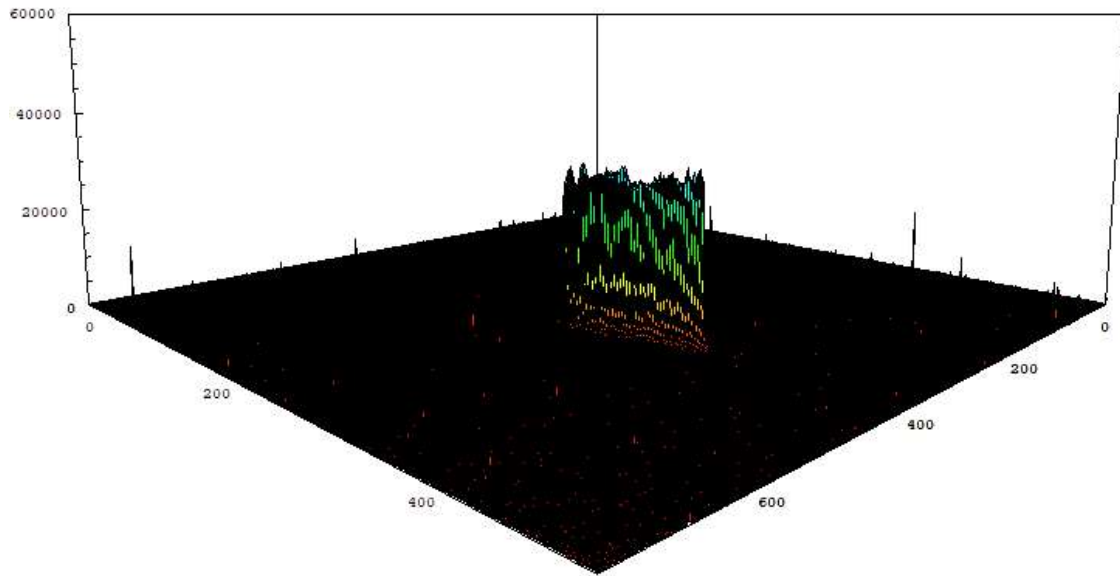
matx = Input["Please enter the x-direction number of pixels in the CCD chip:"]; maty = Input["Please enter the y-direction number of pixels in the CCD chip:"]; Print["CCD chip size (x,y): (", matx, ",", maty, ")"];

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

(* 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 *)

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]

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

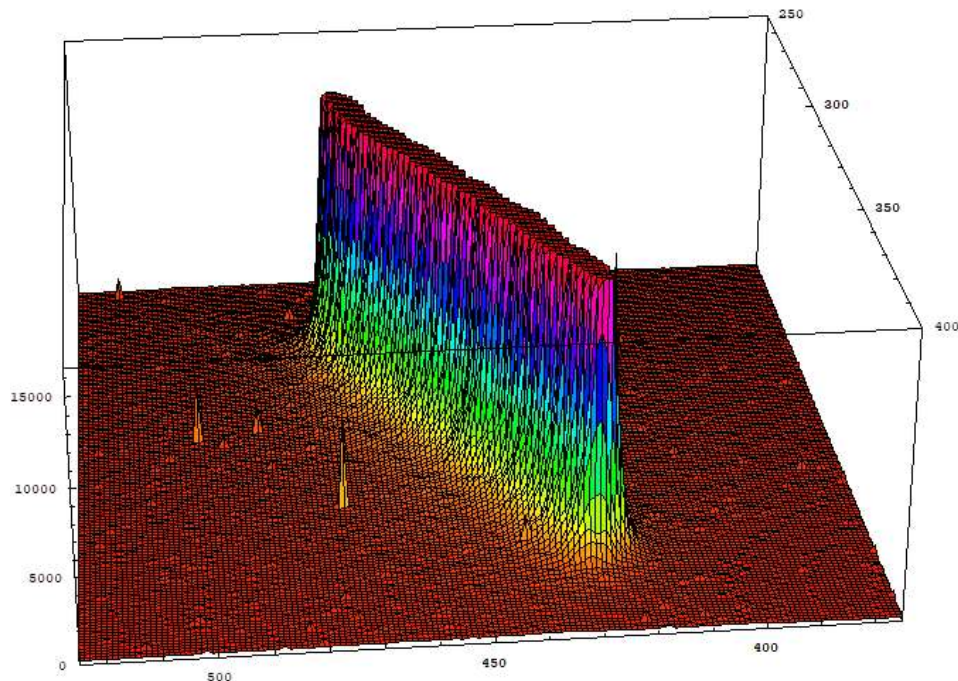


Peak value (ADU): 33159

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

(* The following routine will have to be adjusted until we have a "nice looking" 3D plot of the Polaris path; the data will be chopped (at this moment only for viewing) at half-maximum (mag/2); we usually 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 *)

```
m=N[IntegerPart[mag/2]];ListPlot3D[a,ColorFunction->[Rule]Hue,PlotRange->{{375,525},{250,400},
{0,m}},ViewPoint->[Rule]{.4,4,2}];
```

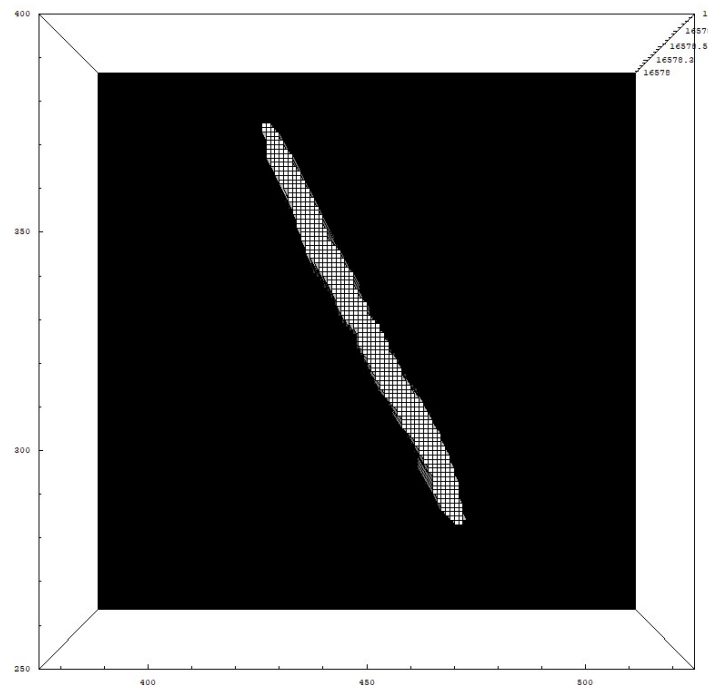


(* Routine 3 - The main data reduction routine *)

(* After the final values x1-x2, y1-y2 are entered, a second viewing of the data, this time from the top, is shown: this will be the standard view in all that follows; and it is also here that the user must indicate: i) the rough main direction of Polaris; ii) the coordinates of the extremes of the path; this can be just an estimate, since there will be a chance to refine this later; all values are printed out as a reminder for later *)

```
nx1=Input["x1?"]; nx2=Input["x2?"]; ny1=Input["y1?"]; ny2=Input["y2?"]; Print["The chosen  
working square goes from ", nx1," to ", nx2," in the x-axis and from ", ny1," to ", ny2," in the y-  
axis"];  
pp2=ListPlot3D[a,ColorFunction->GrayLevel,PlotRange->{{nx1,nx2},{ny1,ny2},{N[m]-1,N[m]}},  
ViewPoint->{0,0,2}]; anspath=Input["Judging by eye, which main direction follows the Polaris path?  
Left-right (1) or Top-down (2)?"];  
If [anspath==1,  
    u=Input["Please indicate the x-coordinate of the left extremity of the path"]; t=Input["Please  
indicate the y-coordinate of the left extremity of the path"]; w= Input["Please indicate the x-  
coordinate of the right extremity of the path"]; v=Input["Please indicate the y-coordinate of the  
right extremity of the path"]; Print["You have picked the path from (",u,"",t,) to  
(",w,"",v,")"],  
    u=Input["Please indicate the x-coordinate of the bottom extremity of the path"]; t=Input["Please  
indicate the y-coordinate of the bottom extremity of the path"]; w=Input["Please indicate the x-  
coordinate of the top extremity of the path"];v=Input["Please indicate the y-coordinate of the top  
extremity of the path"];Print["You have picked the path from (",u,"",t,) to (",w,"",v,")"];  
pathpoints={{u,t},{w,v}};
```

The chosen working square goes from 375 to 525 in the x-axis and from 250 to 400 in the y-axis



You have picked the path from (470,275) to (430,380).

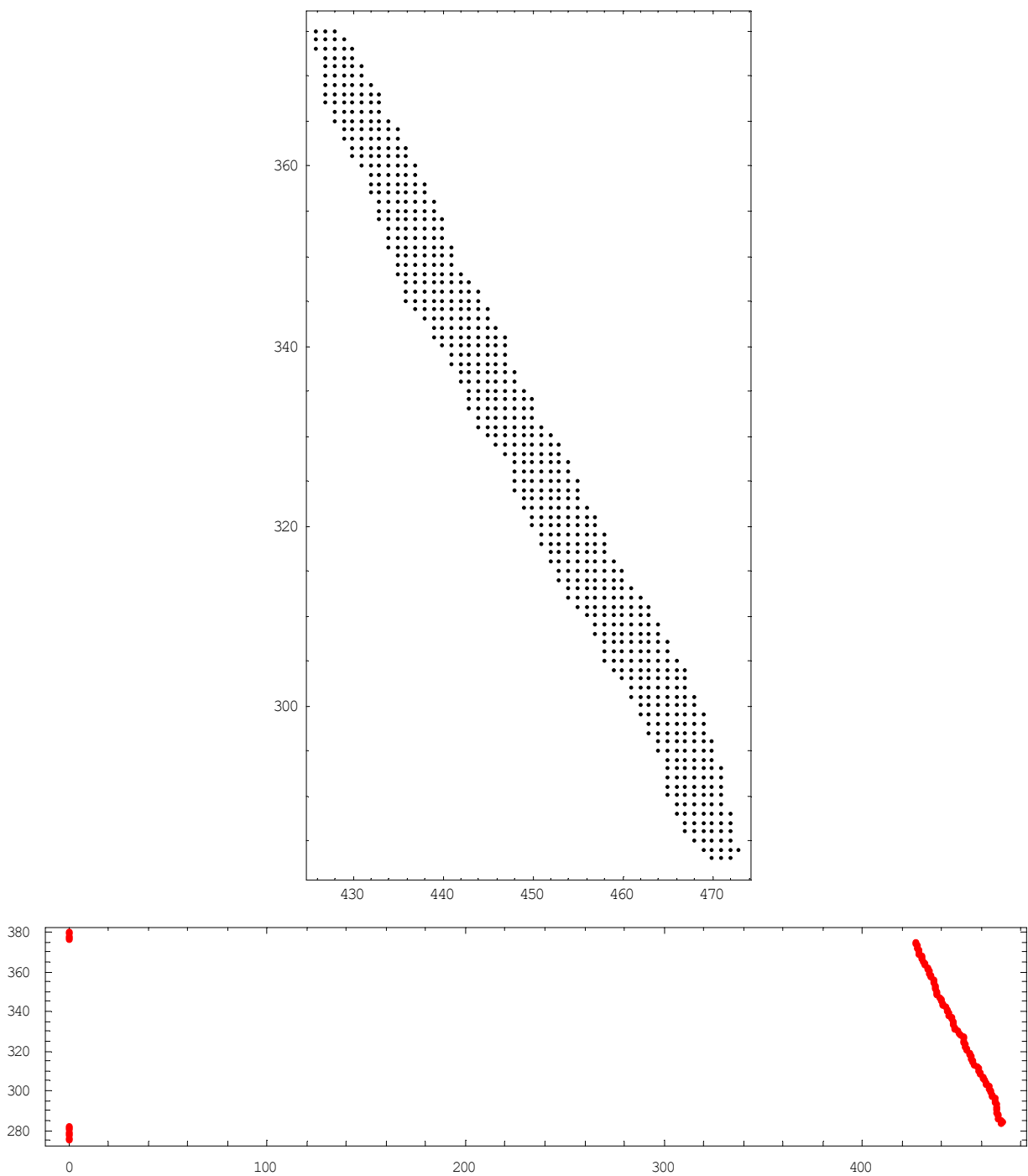
(* The following routine will choose the best axis (xx or yy) to define a path in our chopped data: the one making the smallest angle; it uses the user-entered path extremes; this will also be corrected later, if/when the path extreme points are changed; here, it calculates the angle with each of the xx and yy axis, determining for which this is smallest (no=1 is small with yy; otherwise inver=1 (small with xx)); we also make sure that we always have $t \leq v$ and $u \leq w$ *)

```
b=Transpose[a];
angyy=0;angxx=0;no=0;inver=0;angyy=ArcTan[Abs[u-w]/Abs[t-v]];angxx=Pi/2-angyy;
If[angxx>angyy,
  no=1; If[t>v,alter=t;t=v;v=alter];If[u>w,alter=u;u=w;w=alter],
  inver=1; If[t>v,alter=t;t=v;v=alter];If[u>w,alter=u;u=w;w=alter]];
```

(* Now this routine (actually split into two, depending on the axis chosen) will i) a) if the smallest angle is with the yy axis (no=1), start with the top row (of the top-left pixel identified) and redefine the value of every pixel $\geq m$ to m (new vector d), while all others will be set to zero; at the same time, vector e is initiated, having as first point the midpoint (or the one on its left, if even number of pixels) of the top row; note that if this is not continuous, the midpoint is defined by the set of points where the top-left pixel is located; b) the routine proceeds to the next row, and so on, stopping after concluding the row of the bottom-right pixel above identified; at each row n, the nth pixel of vector e is defined, making sure it will become a continuous vector by the following rule: the pixel in row n is the one closest to the pixel of row n-1 AND to the midpoint in row n, as defined in i (hence, each pixel of vector e is chosen among three possibilities); ii) if the smallest angle is with the xx axis (inver=1), do the same but, this time, column by column, from the pixel at top-left to the pixel at bottom-right (vectors d and e are defined, similarly) *)

(* 1st the subroutine for the case in which the yy-axis is chosen (no=1); several plots in the end [graph plotting included] *)

```
If[no==1,x=u;x1=u;y=t;d={};e={};midp=0;soma=0;midpoint={};i=0;jset=0;
  While[y<=v,
    While[x<matx+1,
      While[b[[x,y]]>=m,d=Append[d,{x,y}];soma=soma+x;x=x+1; i=i+1];
      If[i≠0,
        jset=jset+1; midp=IntegerPart[soma/i];midpoint=Append[midpoint,midp];midp=0;i=0;
        soma=0,
        x=x+1]];
    x2=0;k=1;dist1=0;dist=matx*2;While[k≤jset,dist1=Abs[midpoint[[k]]-x1]; If[dist1<dist,
      x2= midpoint[[k]];dist=dist1;k=k+1];
    If[x1≠u || y≠t, While[x2<x1-1 && b[[x2+1,y]]≥m,x2=x2+1]; While[x2>x1+1 && b[[x2-1,y]]≥m,
      x2=x2-1]; e=Append[e,{x2,y}];x1=x2,
      e=Append[e,{x2,y}];x1=x2];
    y=y+1;x=1;i=0;jset=0;midpoint={}}];
e1=ListPlot[d,PlotRange->All, PlotStyle->PointSize[0.013],Frame->True,Axes->False, AspectRatio->Automatic];
e2=ListPlot[e,PlotRange->All,PlotStyle->RGBColor[1,0,0],PlotStyle->PointSize[0.013],Frame->True,
  Axes->False,AspectRatio->Automatic];
le=Length[e];Print ["Vector e has ",le, " points"];e
```



Vector e has 106 points

{0, 275}, {0, 276}, {0, 277}, {0, 278}, {0, 279}, {0, 280}, {0, 281}, {0, 282}, {470, 283}, {471, 284}, {470, 285}, {469, 286}, {469, 287}, {469, 288}, {468, 289}, {468, 290}, {468, 291}, {468, 292}, {468, 293}, {467, 294}, {467, 295}, {467, 296}, {466, 297}, {466, 298}, {465, 299}, {465, 300}, {464, 301}, {464, 302}, {463, 303}, {463, 304}, {462, 305}, {461, 306}, {461, 307}, {460, 308}, {460, 309}, {459, 310}, {459, 311}, {458, 312}, {457, 313}, {456, 314}, {456, 315}, {455, 316}, {455, 317}, {454, 318}, {454, 319}, {453, 320}, {453, 321}, {452, 322}, {452, 323}, {451, 324}, {451, 325}, {451, 326}, {451, 327},

```
{450,328}, {449, 329}, {448, 330}, {447, 331}, {447, 332}, {446, 333}, {446,334}, {446, 335}, {445, 336}, {445, 337}, {444, 338}, {444, 339}, {443,340}, {443, 341}, {442, 342}, {441, 343}, {441, 344}, {440, 345}, {440,346}, {439, 347}, {438, 348}, {438, 349}, {438, 350}, {437, 351}, {437, 352}, {437, 353}, {436, 354}, {436, 355}, {436, 356}, {435, 357}, {435,358}, {434, 359}, {434, 360}, {433, 361}, {433, 362}, {432, 363}, {432,364}, {431, 365}, {430, 366}, {430, 367}, {430, 368}, {429, 369}, {429,370}, {429, 371}, {428, 372}, {428, 373}, {427, 374}, {427, 375}, {0,376}, {0, 377}, {0, 378}, {0, 379}, {0, 380}
```

(* 2nd the subroutine for the case in which the xx - axis is chosen (inver = 1); several plots in the end [graph plotting not included] *)

```
y = t; y1 = t; x = u; d = {}; e = {}; midp = 0; soma = 0; midpoint = {}; i = 0; jset = 0;
While[x ≤ w,
  While[y < maty + 1, While[b[[x, y]] ≥ m, d = Append[d, {x, y}]; soma = soma + y; y = y + 1; i = i+1];
  If[i ≠ 0, jset = jset + 1; midp = IntegerPart[soma/i]; midpoint = Append[midpoint, midp];
  midp=0; i = 0; soma = 0,
  y = y + 1]];
y2 = 0; k = 1; dist1 = 0; dist = maty*2;
While[k ≤ jset, dist1 = Abs[midpoint[[k]]-y1]; If[dist1 < dist, y2 = midpoint[[k]]; dist = dist1];
  k = k + 1];
If[y1 ≠ t || x ≠ u, While[y2 < y1 - 1 && b[[x, y2 + 1]] ≥ m, y2 = y2 + 1]; While[y2 > y1 + 1 &&
  b[[x, y2-1]] ≥ m, y2 = y2-1]; e = Append[e, {x, y2}]; y1 = y2, e = Append[e, {x, y2}]; y1=y2];
x = x + 1; y = 1; i = 0; jset = 0; midpoint = {}]
```

(* Finally, the iteration routine, showing more plots (namely the Polaris path in red) and allowing to better refine the initial and final points for the path *)

```
pp1 = ListPlot[pathpoints, PlotRange -> All, PlotStyle -> RGBColor[1, 0, 0], PlotStyle ->
PointSize[0.013], Frame -> True, Axes -> False, AspectRatio -> Automatic]; Show[e1, pp1];
Show[pp2]; Show[e1, e2]; anshere = 1; anshere = Input["Please confirm (1) that the path extremes
are correct (red points), comparing with the original plot. If not, write 0."];
While[anshere == 0,
  If[anspath == 1, u = Input["Please indicate the x-coordinate of the left extremity of the path"];
  t = Input["Please indicate the y-coordinate of the left extremity of the path"];
  w = Input["Please indicate the x-coordinate of the right extremity of the path"];
  v = Input["Please indicate the y-coordinate of the right extremity of the path"]; Print["You
  have picked the path from (" , u, ", ", t, ") to (" , w, ", ", v, ")."],
  u = Input["Please indicate the x-coordinate of the bottom extremity of the path"];
  t = Input["Please indicate the y-coordinate of the bottom extremity of the path"];
  w = Input["Please indicate the x-coordinate of the top extremity of the path"];
  v = Input["Please indicate the y-coordinate of the top extremity of the path"]; Print["You have
  picked the path from (" , u, ", ", t, ") to (" , w, ", ", v, ")."]];
pathpoints = {{u, t}, {w, v}}; angyy = 0; angxx = 0; no = 0; inver = 0;
angyy = ArcTan[Abs[u - w]/Abs[t - v]]; angxx = Pi/2 - angyy;
If[angxx > angyy, no = 1; If[t > v, alter = t; t=v; v = alter]; If[u > w, alter = u; u = w; w = alter],
  inver = 1; If[t > v, alter = t; t = v; v = alter]; If[u > w, alter = u; u = w; w = alter]];
If[no == 1, x = u; x1 = u; y = t; d = {}; e = {}; midp = 0; soma = 0; midpoint = {}; i = 0; jset = 0;
```

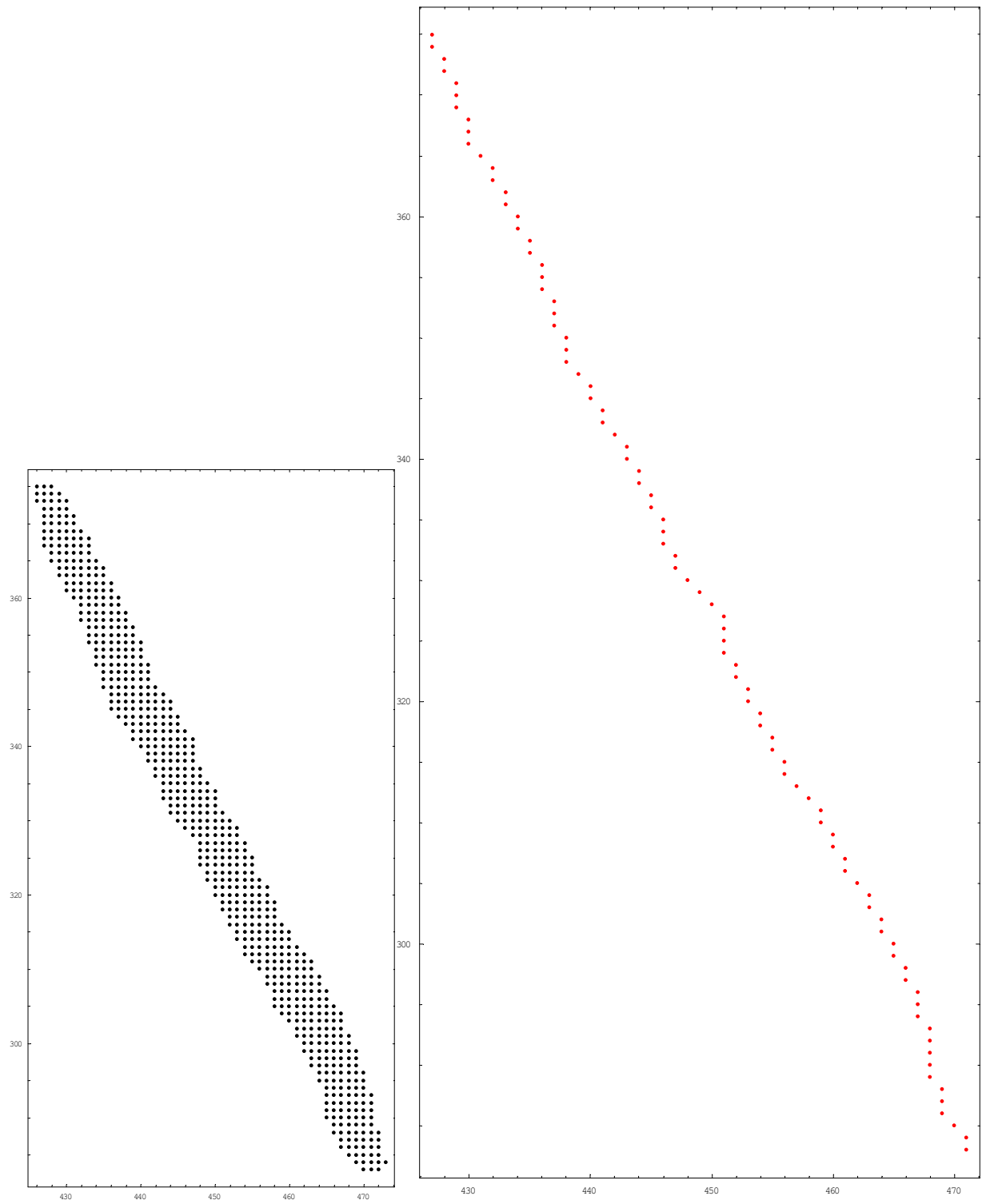
```

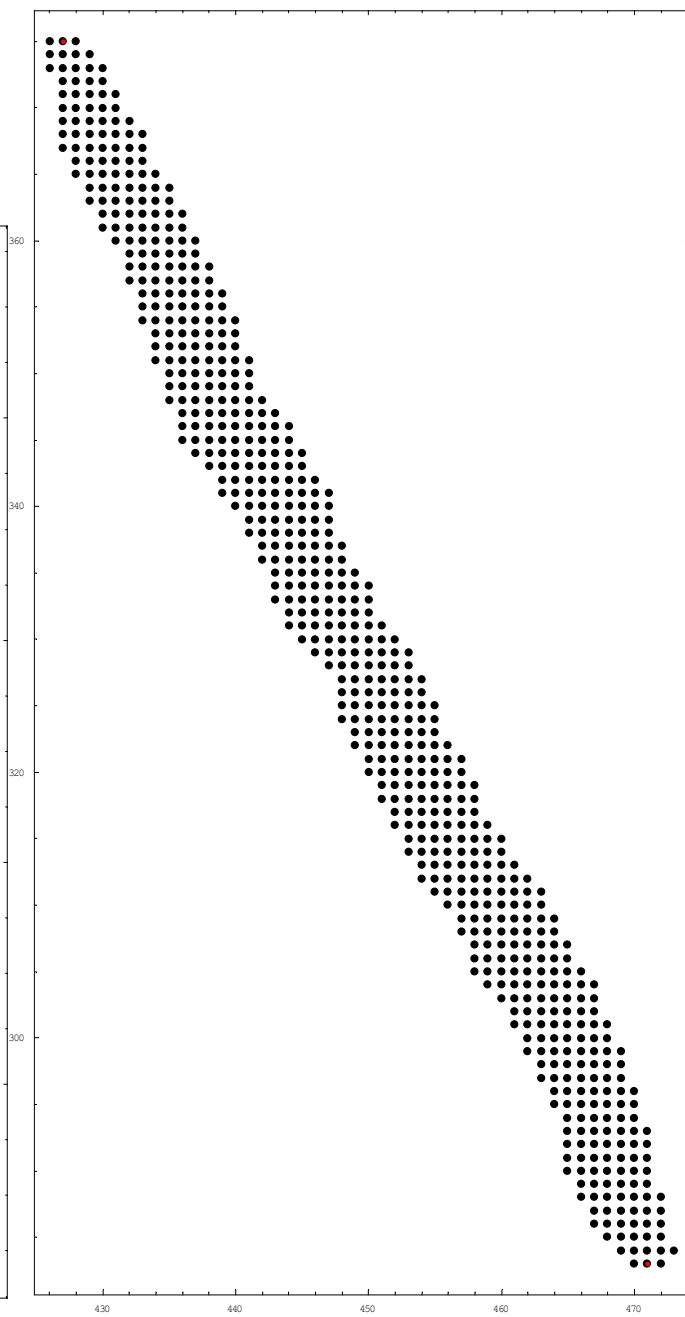
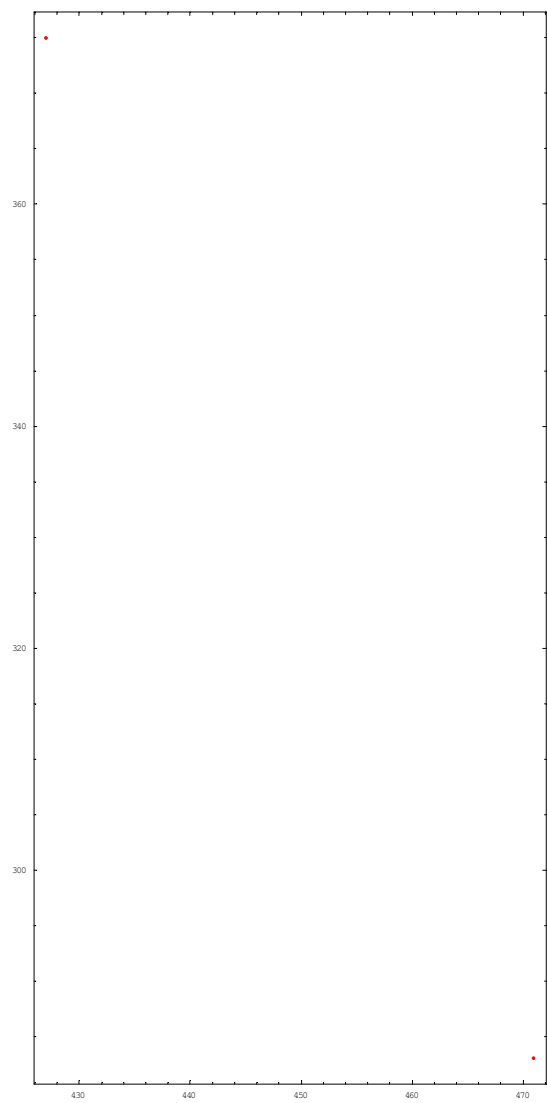
While[y <= v,
  While[x < matx + 1, While[b[[x, y]] >= m, d = Append[d, {x, y}]; soma = soma + x; x = x + 1; i = i + 1];
  If[i ≠ 0, jset = jset + 1; midp = IntegerPart[soma/i]; midpoint = Append[midpoint, midp];
  midp = 0; i = 0; soma = 0,
  x = x + 1]];
x2 = 0; k = 1; dist1 = 0; dist = matx*2; While[k ≤ jset, dist1 = Abs[midpoint[[k]] - x1];
If[dist1 < dist, x2 = midpoint[[k]]; dist = dist1]; k = k + 1];
If[x1 ≠ u || y ≠ t, While[x2 < x1 - 1 && b[[x2 + 1, y]] ≥ m, x2 = x2 + 1]; While[x2 > x1 + 1 &&
b[[x2 - 1, y]] ≥ m, x2 = x2 - 1]; e = Append[e, {x2, y}]; x1 = x2,
e = Append[e, {x2, y}]; x1 = x2];
y = y + 1; x = 1; i = 0; jset = 0; midpoint = {};
y = t; y1 = t; x = u; d = {}; e = {}; midp = 0; soma = 0; midpoint = {}; i = 0; jset = 0;
While[x ≤ w,
  While[y < maty + 1, While[b[[x, y]] >= m, d = Append[d, {x, y}]; soma = soma + y; y = y + 1; i = i + 1];
  If[i ≠ 0, jset = jset + 1; midp = IntegerPart[soma/i]; midpoint = Append[midpoint, midp];
  midp = 0; i = 0; soma = 0,
  y = y + 1]];
y2 = 0; k = 1; dist1 = 0; dist = maty*2; While[k ≤ jset, dist1 = Abs[midpoint[[k]] - y1];
If[dist1 < dist, y2 = midpoint[[k]]; dist = dist1]; k = k + 1];
If[y1 ≠ t || x ≠ u, While[y2 < y1 - 1 && b[[x, y2 + 1]] ≥ m, y2 = y2 + 1]; While[y2 > y1 + 1 &&
b[[x, y2 - 1]] ≥ m, y2 = y2 - 1]; e = Append[e, {x, y2}]; y1 = y2,
e = Append[e, {x, y2}]; y1 = y2];
x = x + 1; y = 1; i = 0; jset = 0; midpoint = {}];];
e1 = ListPlot[d, PlotRange -> All, PlotStyle -> PointSize[0.013], Frame -> True, Axes -> False,
AspectRatio -> Automatic]; e2 = ListPlot[e, PlotRange -> All, PlotStyle -> RGBColor[1, 0, 0],
PlotStyle -> PointSize[0.013], Frame -> True, Axes -> False, AspectRatio -> Automatic];
le = Length[e]; Print ["Vector e has ", le, " points"]; e; pp1 = ListPlot[pathpoints, PlotRange -> All,
PlotStyle -> RGBColor[1, 0, 0], PlotStyle -> PointSize[0.013], Frame -> True, Axes -> False,
AspectRatio -> Automatic]; Show[e1, pp1]; Show[pp2]; Show[e1, e2]; anshere = 1; anshere =
Input["Please confirm (1) that the path extremes are correct (red points), comparing with the
original plot. If not, write 0."];
Print ["Vector e has ", le, " points"]; e

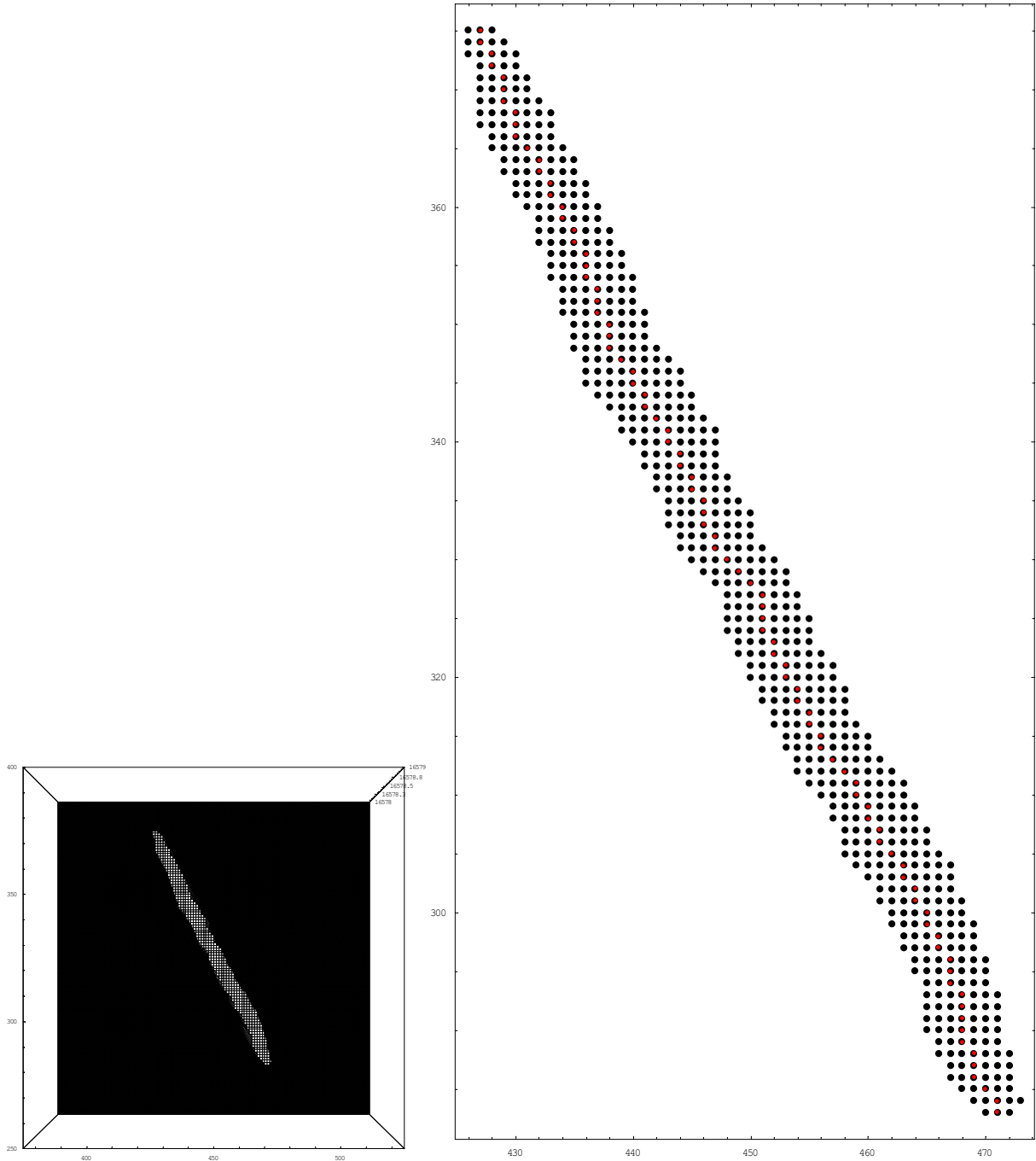
```

[in this demonstration we skip here the output of the several iterations and only show the final graphs in what follows]

You have picked the path from (471,283) to (427,375)







Vector e has 93 points

{471, 283}, {471, 284}, {470, 285}, {469, 286}, {469, 287}, {469, 288}, {468, 289}, {468, 290}, {468, 291}, {468, 292}, {468, 293}, {467, 294}, {467, 295}, {467, 296}, {466, 297}, {466, 298}, {465, 299}, {465, 300}, {464, 301}, {464, 302}, {463, 303}, {463, 304}, {462, 305}, {461, 306}, {461, 307}, {460, 308}, {460, 309}, {459, 310}, {459, 311}, {458, 312}, {457, 313}, {456, 314}, {456, 315}, {455, 316}, {455, 317}, {454, 318}, {454, 319}, {453, 320}, {453, 321}, {452, 322}, {452, 323}, {451, 324}, {451, 325}, {451, 326}, {451, 327}, {450, 328}, {449, 329}, {448, 330}, {447, 331}, {447, 332}, {446, 333},

```
{446, 334}, {446, 335}, {445, 336}, {445, 337}, {444, 338}, {444, 339}, {443, 340}, {443, 341}, {442, 342}, {441, 343}, {441, 344}, {440, 345}, {440, 346}, {439, 347}, {438, 348}, {438, 349}, {438, 350}, {437, 351}, {437, 352}, {437, 353}, {436, 354}, {436, 355}, {436, 356}, {435, 357}, {435, 358}, {434, 359}, {434, 360}, {433, 361}, {433, 362}, {432, 363}, {432, 364}, {431, 365}, {430, 366}, {430, 367}, {430, 368}, {429, 369}, {429, 370}, {429, 371}, {428, 372}, {428, 373}, {427, 374}, {427, 375}}
```

```
(* Routine 4 *)
```

```
(* ONLY if necessary : comment out; note that points of the form (0, y) or (x, 0) will have to be removed from vector e *)
```

```
(* e = {}; e2 = ListPlot[e, PlotRange -> All, PlotStyle -> PointSize[0.013], Frame -> True, Axes -> False, AspectRatio -> Automatic] ; le = Length[e]; Print ["Vector e has ", le, " points"]; e *)
```

```
(* Now begins the SEEING DETERMINATION section *)
```

```
(* Routine 5 (seeing) *)
```

```
(* Routine 5.1 (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, 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 *)
```

```
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]]];lattrstr=StringLength[atrstr];lbtrstr=StringLength[btrstr];
atr2=ToExpression[StringTake[atrstr,{8,lattrstr}]];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,lbtms
tr}]];

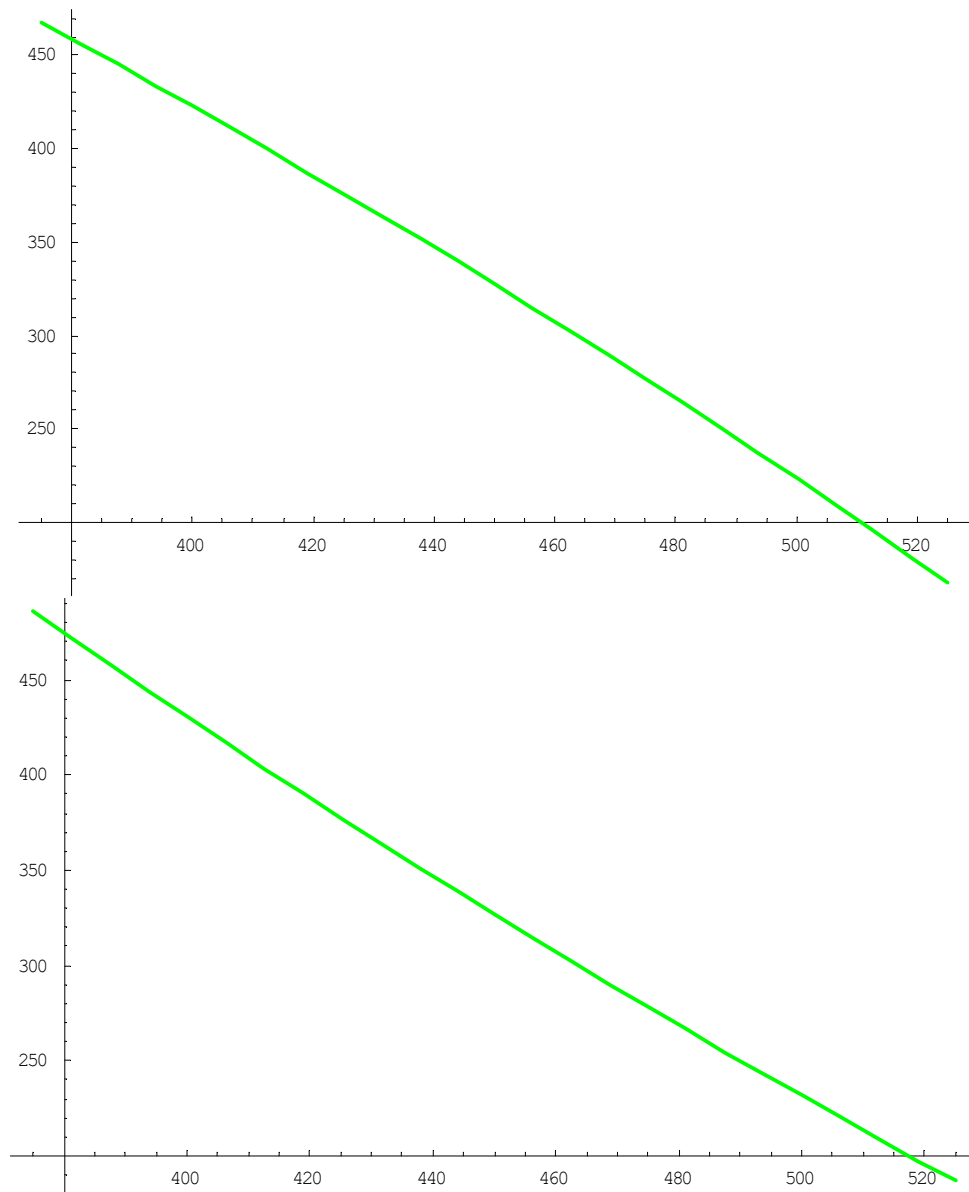
```

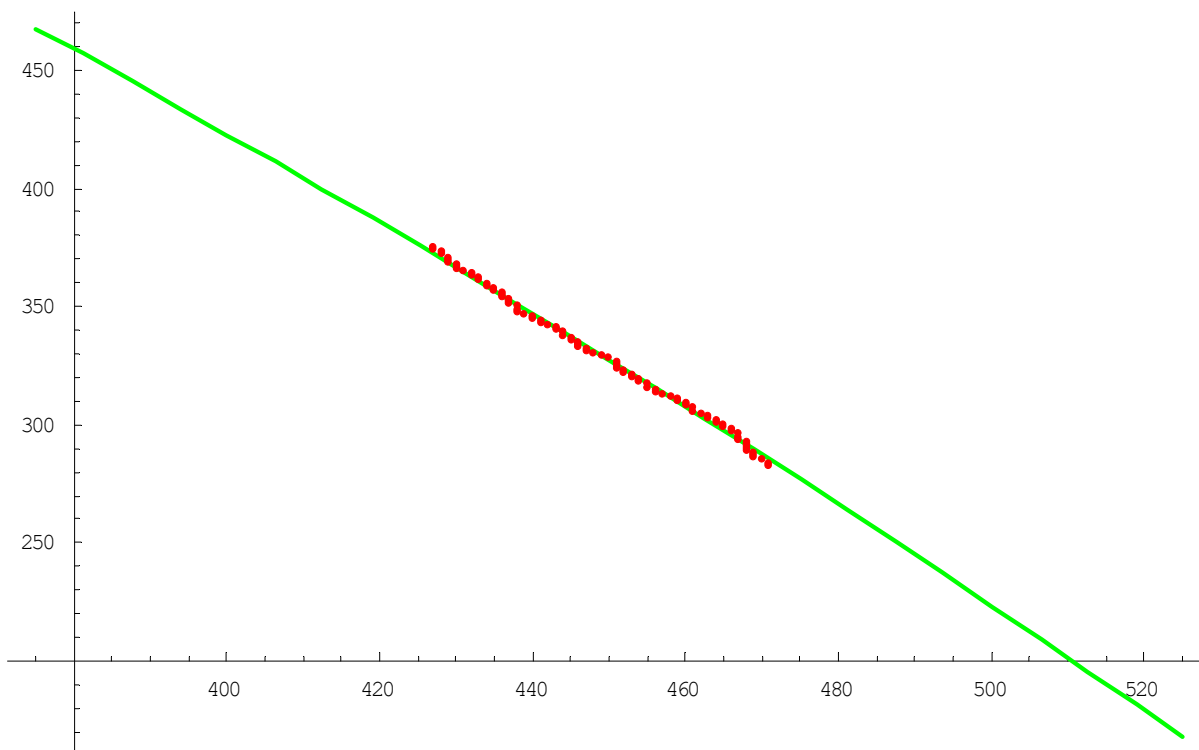
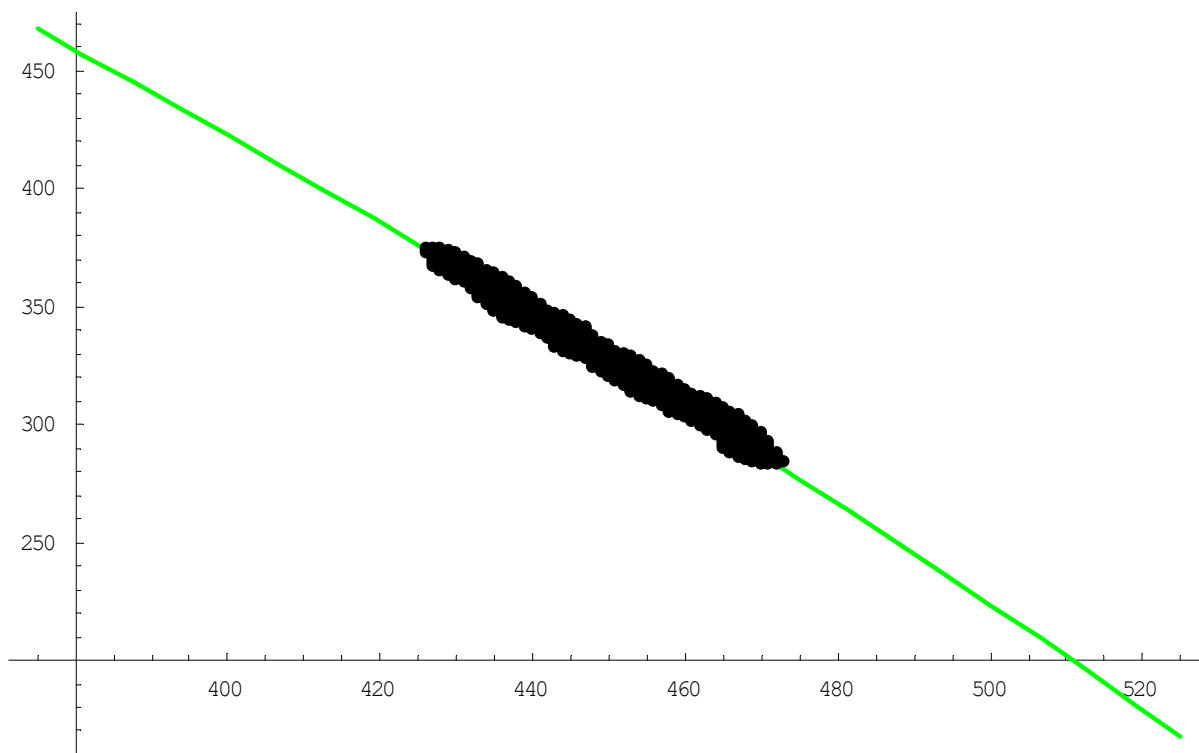
(* We now build each graph fit *)

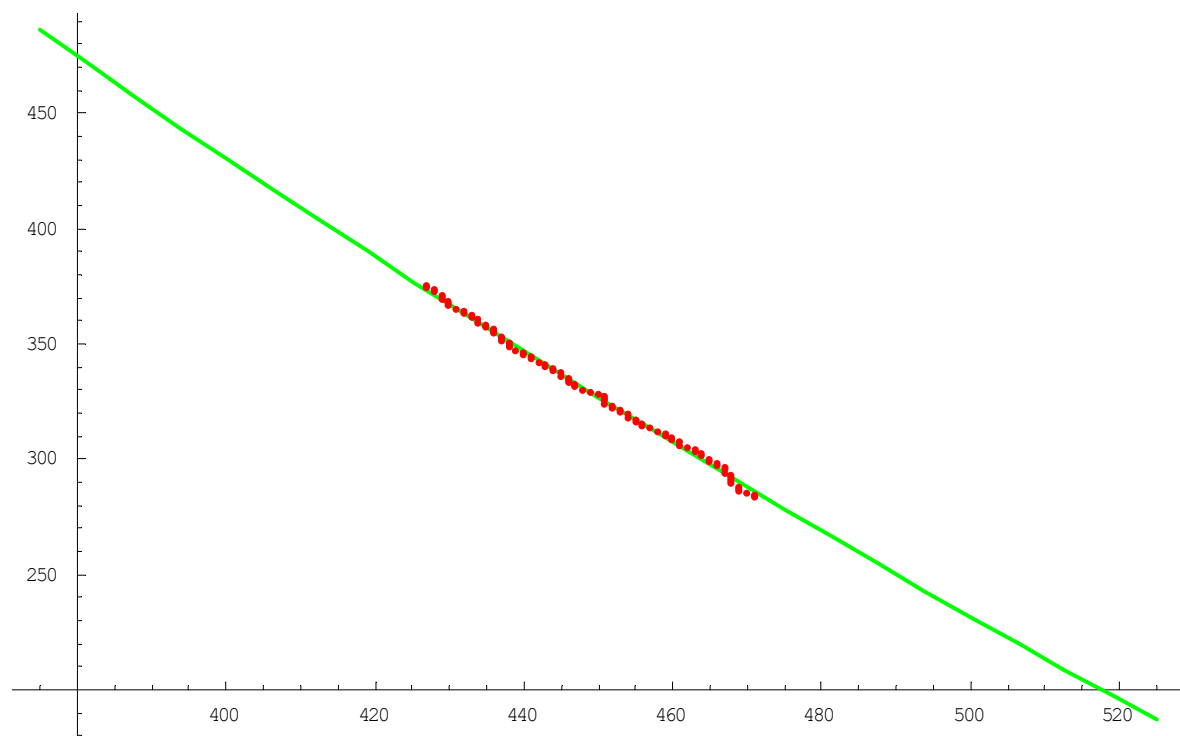
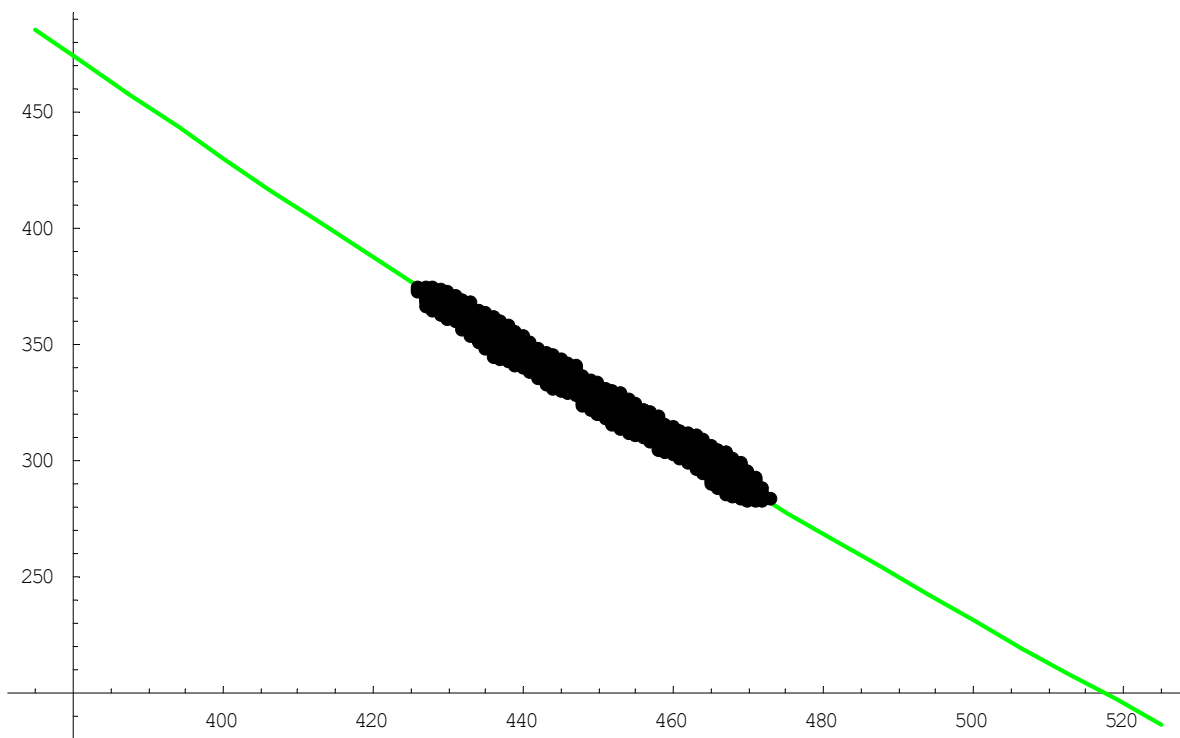
```

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[e10,e2];
Show[e20,e1];Show[e20,e2];

```







(* 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[(ttt-e[[l,1]])^2+(at+sinal*Sqrt[Abs[3233^2-(ttt-bt)^2]]-
e[[l,2]])^2],ttt];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]
```

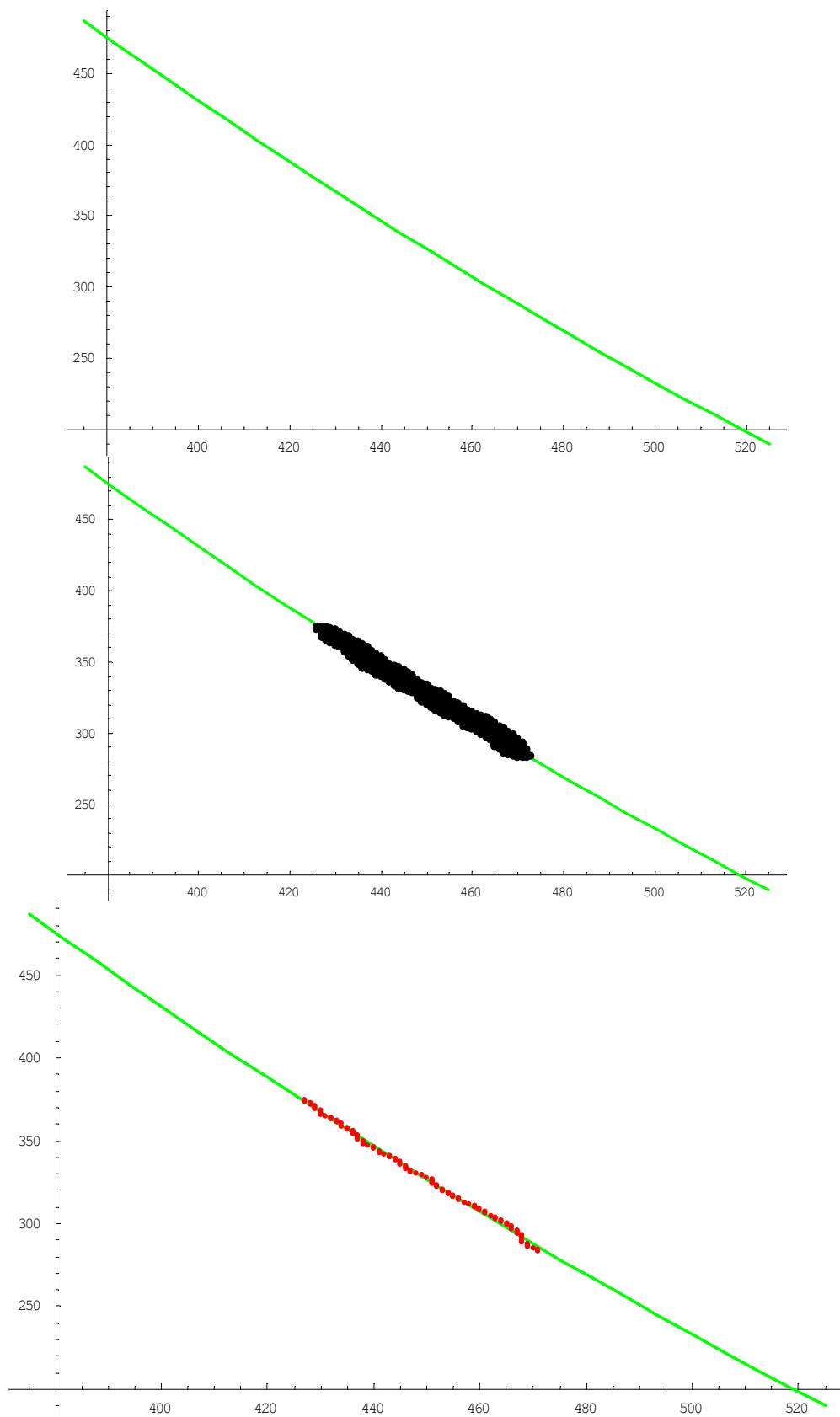
CCD Seeing (R.M.S.) - CIRCLE FIT in arcseconds = 0.528885

(* Routine 5.2 (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[e3, e2]; 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 = 0.499071

(* Routine 5.3 ("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 - (ty - bt)^2];  
While[l ≤ n2, xarcocirculo = Solve[ArcoCirculo == e[[l, 2]], ty]; valor = ty /. xarcocirculo[[1]]; sdf = Abs[valor - e[[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 ];
```