Bull. Spec. Astrophys. Obs., 1997, 43, 164–170

Fast image processing methods for PC: 5. Simultaneous cleaning of the impulse noise in two corresponding frames

Ts.B. Georgiev

Rozhen National Astronomical Observatory, BG-4700 Smolyan, Bulgaria, tsgeorg@phys.acad.bg
Visiting astronomer in the SAO of the RAS

Received April 30, 1997; accepted May 5, 1997.

Abstract.
A fast method and C–program for simultaneous removal of the impulse noise in two corresponding CCD frames is described. For the current pixel the program builds the local histogram of the ratios (or differences) of the values of the corresponding pixels and derives the median and halfwidth (sigma) of the histogram. Applying an appropriate threshold (3–4 sigma) the program recognizes strongly deviated pixel values and improves them by means of the median value and the corresponding pixel value in the other frame. In the regions of stellar images the local histogram is wider than in the background area. For this reason, depending on the user supplied threshold, the procedure removes the images of the cosmic events, but does not change the stellar images.

Key words: photometry – method – noise reduction

1. Introduction

In the previous papers I described four fast methods for astronomical image processing – median filtering, regression smoothing, iterative restoration and iterative cleaning of CCD frames (Georgiev, 1996abcd). The relevant procedures are incorporated in one image processing package for personal computers (Georgiev, 1995), which is based on the software PCVISTA of Treffers and Richmond (1989). The C-texts of the programs, concerning the first three methods, are published. The improved and generalized versions of the programs are available on request. All additional programs work on-line (without intermediate writing the whole frame in the computer memory) and for this reason they are suitable for processing of large frames.

The estimation of the proper signal in the presence of strong impulse noise (cosmic events, defects) is a complicated problem in the astronomical image processing. Two standard methods for frame cleaning (realized in the MIDAS) are described and discussed by Baade and Lucy (1989). The first method, suitable for smoothing of the images of extended objects (galaxies, comets), is the median filtering of the peak pixel values. One fast version of the method is given by Georgiev (1996a, hereafter Paper 1). However this method is not applicable to stellar fields because it cuts the peaks of the star images. The second more sophisticated method, suitable for crowded stellar fields, recognizes and removes sharp peaks using information about the signal and noise of the frame. Unfortunately, this method is too slow.

To find a better practical solution, suitable for small computers, I included one natural improvement of the classical median filtering method, given in Paper 1. The result is described in Part 2 of this paper. In Part 3 I give a description of another fast method which cleans simultaneously two frames. The appropriate C-program (applicable also for processing big frames) is described briefly in Part 4 and its text is included in the Appendix. Some recommendations are given in Part 5.

2. Improvements of the median filtering method described in Paper 1

Obviously, the median filtering method must be improved to omit stellar peaks. For this reason two essential improvements are included in the last version of my program of median processing, available in Paper 1 (hereafter MEDFIL).

Naturally, MEDFIL uses the local frame histogram, constructed from all pixel values V(i,j) inside the circular window with the diameter W, centered

on the pixel being processed. The system of possible windows is shown in Paper 1. MEDFIL derives the median pixel value M, which divides the histogram into two equal parts. Using one constant threshold for recognizing the impulse noise MEDFIL treats the sharp stellar peaks as impulse noise and cuts them off.

The first improvement is that MEDFIL estimates additionally the half-width (sigma) of the local histogram S=(M2-M1)/2, where the values M1 and M2 cut 15 % of the most faint and most bright pixels of the histogram, respectively. If the frame region is occupied by a stellar image, the local histogram is essentially wider than in the case of pure background. Then MEDFIL replaces the current pixel value V by the median M if $|V - M| > C \times S$, where C is the user supplied coefficient. In the case C=0 the program performs usual median smoothing of the frame and in the case $C > 0$ it cleans the impulse noise changing only strongly deviated pixel values. For this reason, and depending on the value of C, the program can remove the cosmic events omitting the stellar images.

The second improvement is that MEDFIL optionally outputs one or two frames – cleaned (or smoothed, when C=0) and residual (the difference between the original and cleaned frames). In the case of cleaning the residual frame appears as a bad pixel map. Having the cleaned and the residual frames the user can perform immediate visual control of the cleaning process by means of blinking the original and residual frames.

3. Comparing and cleaning of two frames

The improved fast median filtering method works very well in the case of oversampled CCD frames, when the full width at half of maximum (FWHM) of the point spread function (PSF) corresponds to 5 or more pixels. However, the case of undersampling is more frequent in astronomical practice. Then each stellar peak looks like impulse noise and the improved method again appears inapplicable. Below I describe the method which is somewhat less sensitive to stellar peaks. This method treats simultaneously two frames and gives another solution to the cleaning problem, both reliable and fast. The previous realizations of such a method for the processing of spectroscopic observations has been announced by Vlasyuk (1993) and Galazutdinov (1995). Below I describe the essential part of the algorithm, developed on the basis of fast median processing and its improvements, described in the previous section of this paper.

Let us compare two corresponding frames by the local histogram of the ratios (or differences) of the corresponding pixel values in the circular window with the diameter W, centered on the current pixel. We call two frames "corresponding" if the positions of the objects in them (p.e. stars, galaxies) coincide. Otherwise the pixel system of one frames must be preliminary offset (or rebinned) to the coordinate system of the other frames. However any rebinning method spreads the energy of each peak of the noise to neighbouring pixels and makes the peak less recognizable. For this reason offsetting at an integer number of pixels (if it is sufficient) is recommended. Note, if the exposures and the filters of the frames coincide, it is natural to compare frames using the differences between the corresponding pixels. However, in other, more frequent cases of different exposures or filters we must use the ratio of the pixel values. Though, the frames must be comparable.

Let now m be the median of the local histogram of the ratios (or differences) of the corresponding pixel values and $s = (m2 - m1)/2$ is the estimate of the halfwidth (sigma) of the local histogram, derived in the same way as described in the previous section of this paper, where the values m1 and m2 cut 15 % of the faintest and the brightest pixel values. Then the program recognizes the presence of impulse noise in one of the frames if the current pixel ratio (or difference) values v is deviated from the median m too much, i.e if $|v - m| > C \times s$. As earlier, C is the user supplied coefficient. After this the program must find the place of the peak noise – in the first or in the second frame. For this reason it processes additionally a 3×3 pixels area centered on the current pixel and estimates the sigma values S1, S2 of the pixel distributions in each of the frames. Then the program compares the absolute values of the weighted individual deviations W1=V1/S1 and W2=V2/S2, where V1 and V2 are the values of the current pixels, and decides which pixel is more deviated. At the end the program estimates the current pixel value using the median m and the corresponding pixel value in the other frame and replaces the noised value by its estimate.

Practically the diameter W of the cleaning window must not be too large. When the FWHM of the PSF is about 3 pixels, I prefer W=5. Then one single noise value will be easily recognized not only in the flat (background) area of the image but also in the periphery of a stellar image. However, because of the presence of the wide range of pixel values, the local histogram is wider in the stellar image region than in the background areas. For this reason the peaks of the stellar images will not be considered as impulse noise and they will be saved without changes.

4. The program MDCC

The described method of comparing and cleaning two frames is realized as one C-program called MDCC.

Its text is given in the Appendix. The current version of MDCC processes frames with widths up to 1024 pixels, but the size parameters in the program may be increased. The frames must be presented in integer FITS, in the range -32768 – 32767.

Usually each call of the program MDCC uses two input (raw) frames and produces two output (cleaned) frames. Then the input parameter N must be equal to 2. If N=1, the program will clean only the first frame and the second output frame will be the residual frame, containing the pixel values removed from the first frame. In the case of rebinning the rebinned frame can be used as the second comparison frame in the cleaning process. The input parameter M (mode) defines the method of comparing — using the ratios (M=1) or the differences (M=2) of the corresponding pixels. The window size W and the threshold coefficient C are also input parameters of MDCC. Experience of the author with MDCC, realized as addition to the PCVISTA of Treffers and Richmond (1989), has shown that it removes successfully all significant single impulses. It is recommended that no more than 1 per cent of the pixels in the frame should be changed by MDCC. The hardest cosmics, scattered over some pixels, must be removed preliminary by other tools.

5. Final remarks

Some recommendations about the frame cleaning are given below. The preliminary processing of the CCD frames is concerned with bias and dark subtracting and flat–fielding. A high signal–to–noise ratio of the bias, dark and flat–field frames is very desirable. In these cases the construction of "master" dark–field and flat–field from 3 or more frames, using the median of the corresponding pixels, is strongly recommended (see p.e. Newberry, 1991).

The preparatory processing of the CCD frames concerns the cleaning of the impulse noise, rebinning and adding of the frames. Three cleaning procedures — the median processing of a single frame (Part 2), median comparing and cleaning of two frames (Part 3) and iterative cleaning of a single frame (Georgiev, 1996c) may be used in the above mentioned sequence. However the method given in the present paper, which is realized in the program MDCC, is the most powerful.

It is important to point out that the median processing method, if it is so fast as in the program MEDFIL, ensures good possibilities for the processing of residual frames instead of original ones. I will explain this point of view below. Any cleaning procedure needs visual control, at least in the process of choice of the parameters. If, p.e., the peaks of bright stars happen to be destroyed, the user must increase the corresponding threshold and repeat the cleaning process. However, the best way of analysis of the structure and the noise reduction in a complicated image of an extended object is the visual inspection of the corresponding residual frame. Moreover, my experience shows that all cleaning procedures work on residual frames better than on original ones. For these reasons in the case of complicated objects or crowded stellar fields I recommend cleaning residual frames. It is easy to decompose any original frame into smoothed and residual components (frames) using the program MEDFIL (or the like) with W=25 and C=0. After cleaning, the original frame can be reconstructed (recomposed) by addition of the cleaned residual frame to the smoothed frame.

Acknowledgements. I express gratitude to Dr. R. Treffers and Dr. M. Richmond, for making available the source texts of PCVISTA, to Dr. V.V. Vlasyuk for useful discussions, and to the administration of the SAO for the hospitality and support. This paper is a part of the investigations supported by grant F-700/97 of the Ministry of Education, Science and Technology of Bulgaria.

References

Baade D., Lucy L.B., 1989, Proc. of 1st ESO/ST-ECF Data Analysis Workshop, April 17-19, 1989, Garching, 169

Georgiev Ts.B., 1995, IAU Comission 9, WGWFI Newsletter, **8**, 27

Georgiev Ts.B., 1996a, Bull. Spec. Astrophys. Obs., **39**, 124 (Paper 1)

Georgiev Ts.B., 1996b, Bull. Spec. Astrophys. Obs., **39**, 131

Georgiev Ts.B., 1996c, Bull. Spec. Astrophys. Obs., **39**, 140

Georgiev Ts.B., 1996d, Bull. Spec. Astrophys. Obs., **41**, 140

Galazutdinov G.A., 1995, Ph.D. Thesis, SAO Nizhnij Arkhyz

Newberry M.V., 1991, Publ. Astr. Soc. Pacific, **103**, 122

Treffers R.R., Richmond M.W., 1989, Publ. Astr. Soc. Pacific, **101**, 725

Vlasyuk V.V., 1993, Bull. Spec. Astrophys. Obs., **36**, 107

MIDAS, User's Manual, 1995

```
/*-----------------------------------------------------------------
  MDCC: Median Comparing & Cleaning of two frames, v.2.0, Dec 96
  Tsvetan Georgiev, Rozhen National Astronomical Observatory,
  BG-4700 Smolyan, Bulgaria,  tsgeorg@phys.acad.bg
-----------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
        #include ''pcvista.h''
        #include ''fits.h''
#define   MNC  1024       /* max image width in pixels */
#define   MFW  11     /* max window diameter in pixels */
#define   MRB  32     /* number of rows in the buffers */
#define   MHS  32767              /* max histogram size */
        #ifdef PROTO
        void main(int, char **);
#endif
void main (argc, argv)
int argc;  char *argv[]; {
char *gotit; FITS_HANDLE fi1,fi2,fo1,fo2; FILE *fdat;
static int huge mi1[MRB][MNC];               /* input buffer 1 */
static int huge mi2[MRB][MNC];               /* input buffer 2 */
static int huge m1[MFW][MNC+MFW-1];         /* band for image1 */
static int huge m2[MFW][MNC+MFW-1];         /* band for image2 */
static int huge m[MFW][MNC+MFW-1];      /* band i1/i2 or i1-i2 */
static int huge mo1[MRB][MNC];              /* output buffer 1 */
static int huge mo2[MRB][MNC];              /* output buffer 2 */
static int huge h[MHS];      /* histogram in the median window */
static int r[MNC],pn[MFW];      /* row pos.num in memory m[][] */
static int lw[MFW/2+1];  /* limits of the circle median window */
static int l1[50],l2[50],incr,lt;       /* for ''SHELL'' sorting */
int i,ii,iinp,iout,j,jj,k,l,ll,lk,np,npc,rr,nrbuf=MRB,n,n2;
int mod,nim,nr,nc,nr1,nc1,fw,hw,zp,nh,npix,nib,nob;
int sum,med,hlf,su1,su2,me1,me2,q1,q2,iib,iob,i1,i2,me01,me02;
double csig,sig,s,sd=0.,sdd=0.,ssig=0.,sssig=0.,dif,dif1,dif2;
double dp,d,d1,d2,ec,dnh,p11,p12,p21,p22,dmin,dmax;
long ld,lnh,lzp,nco11=0,nco12=0,nco21=0,nco22=0,lmin,lmax,lneq=0;
  lmin=-MHS; lmax=MHS; dmin=(double)lmin; dmax=(double)lmax;
  nh=MHS-2; dnh=(double)nh; lnh=(long)nh;  /* hist.range: 1-nh */
  nim=2;  mod=2;  fw=5;  csig=3.5; ec=5000.; lzp=zp=16000;
if (argc < 5)  {  Usage:
printf(''MDCC  inp1  inp2  out1  out2  [n=N m=M w=W c=C]\n'');
printf(''Median comparing & cleaning the inpulse noise in 1 or 2\n'');
printf(''  respective images inp1,inp2;   v.2.0, Dec 96\n'');
printf(''N=1/2 - number of cleaned images, DEFAULT N=%d;\n'',nim);
printf(''  N=1 - cleaning only imp1, when inp2 is 'better',\n'');
printf(''  results: out1 is cleaned inp1,  out2=inp1-out1;\n'');
printf(''  N=2 - cleaning inp1 & inp2, results; out1 & out2;\n'');
printf(''M=1/2 - kind of frame comparing,  DEFAULT M=%d;\n'',mod);
printf(''  M=1 - using the ratio of the respective pixel values,\n'');
printf(''  for frames with diff.mean signals, if inp1/inp2<6;\n'');
printf(''  M=2 - using the defference between the pixel values,\n'');
printf(''  for frames with almost equal mean signal;\n'');
printf(''W=3/4/5..%d - window diameter,    DEFAULT W=%d;\n'',MFW,fw);
printf(''C=3-5 - sigma coefficient,        DEFAULT C=%3.1f\n'',csig);
printf(''After each execution MDCC writes an info-file 'mdcc.jrn'');
return; }
if(argc>5) {gotit=find(''n'',argv[5]); nim=(int)(evaluate(gotit));}
```

```
  if(nim!=1&&nim!=2) { printf(`` - N? -\n''); goto Usage; }
if(argc>6) {gotit=find(``m'',argv[6]); mod=(int)(evaluate(gotit));}
  if(mod<1&&mod>2) { printf(`` - M? -\n''); goto Usage; }
if(argc>7) {gotit=find(``w'',argv[7]); fw=(int)(evaluate(gotit));}
  if(fw<3||fw>MFW) { printf(`` - W? -\n''); goto Usage; }
if(argc>8) {gotit=find(``c'',argv[8]); csig=(evaluate(gotit));}
/* ---------------------- PRELIMINARY PART ---------------------- */
hw=fw/2;                        /* half-width of the smooth.window */
if (hw*2==fw) rr=hw*hw;   else rr=(int)((hw+0.5)*(hw+0.5));
npix=1; for (k=0; k<=hw; k++) { lw[k]=0;        /* window limits */
for(l=0;l<=hw;l++) if(k*k+l*l<=rr) {lw[k]=l; if(k>0) npix+=4;}}
fw=hw*2+1;  hlf=npix/2+1;          /* half of the histogram total */
q1=(int)((double)npix*0.15+.5);   /* left  quantil limit of hist */
q2=(int)((double)npix*0.85+.5);   /* right quantil limit of hist */
/* ------------------------ opening files ---------------------- */
fi1=fits_open(argv[1],``r'',&nr,&nc);
if(nc>MNC) { printf(`` - Too wide inp1 -\n''); goto Usage; }
fi2=fits_open(argv[2],``r'',&nr1,&nc1);  if(nr1!=nr||nc1!=nc) {
printf(`` - Different sizes of inp1 & inp2 -\n''); goto Usage; }
fo1=fits_open(argv[3],``w'',&nr,&nc);
fo2=fits_open(argv[4],``w'',&nr,&nc);  dp=(double)nr*(double)nc;
printf(``%dx%d=%1.0f n=%d m=%d w=%d,%dp c=%3.1f '',
nr,nc,dp,nim,mod,fw,npix,csig);
/* printf(``nrb=%d zp=%d ec=%3.0f q1=%d hlf=%d q2=%d '',
nrbuf,zp,ec,q1,hlf,q2); */
/* ---------------- MAIN LOOP ALONG THE IMAGE ROWS ------------- */
for (iinp=0; iinp<nr+2*hw; iinp++) {                  /* i-LOOP */
/* ---- check the row number and filling the input buffers ----- */
nib=iinp/nrbuf;             /* number of the current input buffer */
iib=iinp-nib*nrbuf;    /* input row position in the input buffer */
if (iib==0) { i1=nib*nrbuf-hw; i2=i1+nrbuf; if(i2>nr+hw) i2=nr+hw;
  for (i=i1; i<i2; i++) { ii=i;          /* GET BAND FROM IMAGE 1 */
if(i<0) ii=nr+i; if(i>=nr) ii=i-nr;    /* for upper/down margins */
fits_get_data(fi1,ii,0,r,nc); for(j=0;j<nc;j++) mi1[i-i1][j]=r[j];}
  for (i=i1; i<i2; i++) { ii=i;          /* GET BAND FROM IMAGE 2 */
if(i<0) ii=nr+i; if(i>=nr) ii=i-nr;    /* for upper/down margins */
fits_get_data(fi2,ii,0,r,nc); for(j=0;j<nc;j++) mi2[i-i1][j]=r[j];}
/* printf(``\niinp,nib, i1,i2:  %d %d  %d %d'',iinp,nib,i1,i2); */ }
/* ----- check the row number and filling the frame bands ------ */
  for(k=1; k<fw; k++) pn[k-1]=pn[k];  /* rotation of the numbers */
  np=iinp-iinp/fw*fw; pn[fw-1]=np;     /* pos.number of curr.row */
  npc=pn[hw];  /* pos.number of curr.out.row for eventual output */
for (j=0; j<nc; j++) { m1[np][hw+j]=mi1[iib][j];
       m2[np][hw+j]=mi2[iib][j]; }
  if(mod==1) for (j=hw; j<nc+hw; j++) {     /* GET RATIO IM1/IM2 */
d=(double)m1[np][j]/(double)m2[np][j]*ec;      /* check & fill m */
if(d<dmin) d=dmin;  if(d>dmax) d=dmax;  m[np][j]=(int)d; }
  if(mod==2) for (j=hw; j<nc+hw; j++) {/* GET DIFFERENCE IM1-IM2 */
ld=(long)m1[np][j]-(long)m2[np][j]+lzp;        /* check & fill m */
if(ld<lmin) ld=lmin;  if(ld>lmax) ld=lmax;  m[np][j]=(int)ld; }
  for(j=0;j<hw;j++) {       /* ADDING PERIPHERY TO WORKING BANDS */
m1[np][j]=m1[np][nc-hw+j];  m1[np][nc+hw+j]=m1[np][hw+j];
m2[np][j]=m2[np][nc-hw+j];  m2[np][nc+hw+j]=m2[np][hw+j];
m[np][j] = m[np][nc-hw+j];  m[np][nc+hw+j] = m[np][hw+j]; }
  iout=iinp-2*hw;                          /* output row number */
  if(iout<0) goto Next;                   /* preliminary reading */
nob=iout/nrbuf;                       /* num of the output buffer */
```

```
iob=iout-nob*nrbuf;   /* num of the out.row in the output buffer */
/* ---------------- LOOP IN THE CURRENT IMAGE ROW -------------- */
  for (j=hw; j<nc+hw; j++) {
if (j==hw) {           /* FIRST PIXEL IN THE CURRENT ROW */
for (n=0; n<MHS; n++) h[n]=0;        /* histogram initialization */
for (k=0; k<fw; k++)  { np=pn[k];  lk=lw[abs(k-hw)];
for (l=hw-lk; l<=hw+lk; l++) { n=m[np][l];  h[n]++;  } } /* HIST */
med=0; sum=0; while(sum<hlf) { med++; sum += h[med]; } /* MEDIAN */
me1=med-1; su1=sum-h[med]; while(su1>q1) { su1-=h[me1]; me1--; }
me2=med+1; su2=sum+h[me2]; while(su2<q2) { me2++; su2+=h[me2]; } }
else {                 /* OTHER PIXELS IN THE CURRENT ROW */
for (k=0; k<fw; k++) {     /* along the window edge columns only */
np=pn[k];  lk=lw[abs(k-hw)];
  n = m[np][j-lk-1]; h[n]--;  if(n<=med) sum--;      /* removing */
  n = m[np][j+lk];   h[n]++;  if(n<=med) sum++; }     /* adding */
while(sum >=hlf) { sum-=h[med]; med--; }      /* BACK FOR MEDIAN */
while(sum < hlf) { med++; sum+=h[med]; }              /* MEDIAN */
me1=med-1; su1=sum-h[med]; while(su1>q1) { su1-=h[me1]; me1--; }
me2=med+1; su2=sum+h[me2]; while(su2<q2) { me2++; su2+=h[me2]; } }
  sig=(double)(me2-me1)/2.;  /* estimated sigma of the histogram */
  dif=(double)m[npc][j];          /* current difference or ratio */
  if(mod==1) d=dif/ec;  if(mod==2) d=dif-(double)zp;
  sd+=d;  sdd+=d*d;     /* statistics of the ratio or difference */
  ssig+=sig; sssig+=sig*sig;              /* statistics of sigma */
/* --------- ANALYSIS AND REMOVING OF THE IMPULCE NOISE -------- */
dif1=(double)med-csig*sig; dif2=(double)med+csig*sig;  /* limits */
  mo1[iob][j-hw]=m1[npc][j];        /* rewriting without changes */
  if(nim==1) mo2[iob][j-hw]=0; else mo2[iob][j-hw]=m2[npc][j];
if(dif<dif1||dif2<dif) {  /* get medians in 3x3 pix in im1 & im2 */
 np=pn[hw]; me01=m1[np][j]; me02=m2[np][j]; /* respective pixels */
n=0; for (k=hw-1; k<=hw+1; k++) { np=pn[k];          /* get data */
for (l=j-1; l<=j+1; l++) { n++; l1[n]=m1[np][l]; l2[n]=m2[np][l]; } }
  incr=1;  while (incr<n/9) incr=3*incr+1;   /* ''SHELL''sorting 1 */
  while (incr>=1) {  for (jj=incr+1; jj<=n; jj++) { ll=jj-incr;
  while (ll>=1&&l1[ll]>l1[ll+incr]) { lt=l1[ll+incr];
  l1[ll+incr]=l1[ll]; l1[ll]=lt; ll-=incr; } } incr/=3; } n2=n/2;
 s=(double)l1[n2-1]+(double)l1[n2]+(double)l1[n2+1];   /*average */
 me1=(int)(s/3.+0.5);                          /* loc. med1 in 3x3 */
 sig=(double)(l1[7]-l1[2]);                    /* loc.sigma in 3x3 */
 sig+=(double)(l1[8]-l1[3]); sig/=4.;  if(sig<1.) d1=0.;   else
 d1=(double)(m1[npc][j]-me1)/sig;    /* wheigh.dev. from loc.med.*/
  incr=1;  while (incr<n/9) incr=3*incr+1;   /* ''SHELL''sorting 2 */
  while (incr>=1) {  for (jj=incr+1; jj<=n; jj++) { ll=jj-incr;
  while (ll>=1&&l2[ll]>l2[ll+incr]) { lt=l2[ll+incr];
  l2[ll+incr]=l2[ll]; l2[ll]=lt; ll-=incr; } } incr/=3; } n2=n/2;
 s=(double)l2[n2-1]+(double)l2[n2]+(double)l2[n2+1];   /*average */
 me2=(int)(s/3.+0.5);                          /* loc. med2 in 3x3 */
 sig=(double)(l2[7]-l2[2]);                    /* loc.sigma in 3x3 */
 sig+=(double)(l2[8]-l2[3]); sig/=4.;  if(sig<1.) d2=0.;   else
 d2=(double)(m2[npc][j]-me2)/sig;    /* wheigh.dev. from loc.med.*/
/* -------- correction of the inpulse noise in image 1 --------- */
if (fabs(d1)==fabs(d2))  lneq++;
if (fabs(d1)>fabs(d2)) { if(d1<0.) nco11++; else nco12++;
 /* if(mod==1) ld=(long)((double)me2*(double)med/ec); using med2 */
    if(mod==1) ld=(long)((double)me02*(double)med/ec);
else ld=(long)me2+(long)med-lzp;
if(ld<lmin) ld=lmin; if(ld>lmax) ld=lmax; mo1[iob][j-hw]=(int)ld; }
```

```
/* -------- correction of the inpulse noise in image 2 --------- */
if (fabs(d2)>fabs(d1)) { if(d2<0.) nco21++; else nco22++;
if(nim==1) mo2[iob][j-hw]=m1[npc][j]-mo1[iob][j-hw]; if(nim==2) {
 /* if(mod==1) ld=(long)((double)me1/(double)med*ec); using med1 */
    if(mod==1) ld=(long)((double)me01/(double)med*ec);
else ld=(long)me1-(long)med+lzp;
if(ld<lmin) ld=lmin; if(ld>lmax) ld=lmax; mo2[iob][j-hw]=(int)ld; } }
 }   /* ELSE-(changing)-end */          }     /* j-END */
/* ------------------- writing the output buffers -------------- */
if (iob==nrbuf-1 || iout==nr-1) {
for (l=0; l<=iob; l++) { for(j=0;j<nc;j++) r[j]=mo1[l][j];
k=nob*nrbuf+l; fits_put_data(fo1,k,0,r,nc); }
for (l=0; l<=iob; l++) { for(j=0;j<nc;j++) r[j]=mo2[l][j];
k=nob*nrbuf+l; fits_put_data(fo2,k,0,r,nc); } }
Next: if(iout/100*100==iout) printf(''%d '',iout);  }     /* i-END */
/* -------------------------- FINAL PART ---------------------- */
fits_close(fi1); fits_close(fi2); fits_close(fo1); fits_close(fo2);
ssig/=dp; sd/=dp;
sssig=sqrt(sssig/dp-ssig*ssig); sdd=sqrt(sdd/dp-sd*sd);
  if(mod==1) printf(
''\nim1/im2=%4.3f+-%4.3f  hist.sig=%4.3f+-%4.3f  eq.pairs=%ld'',
sd,sdd,ssig/ec,sssig/ec,lneq);
  if(mod==2) printf(
''\nim1-im2=%3.2f+-%3.2f  hist.sig=%2.1f+-%2.1f  cor.pairs=%ld'',
sd,sdd,ssig,sssig,lneq);
  p11=(double)nco11/dp;   p21=(double)nco21/dp;
  p12=(double)nco12/dp;   p22=(double)nco22/dp;  printf(
''\nnum.cuts: im1-%ld+%ld (%4.3f %4.3f)  im2-%ld+%ld (%4.3f %4.3f)'',
  nco11,nco12,p11,p12,nco21,nco22,p21,p22);
fdat = fopen (''mdcc.jrn'',''w'');   fprintf(fdat,
''mdcc:  %s   %s   %s   %s \n'',argv[1],argv[2],argv[3],argv[4]);
fprintf(fdat,''%dx%d=%1.0f n=%d m=%d w,n=%d,%d s=%3.1f '',
nr,nc,dp,nim,mod,fw,npix,csig);
fprintf(fdat,''zp=%d ec=%3.0f q1=%d hlf=%d q2=%d'',zp,ec,q1,hlf,q2);
  if(mod==1) fprintf(fdat,
''\nim1/im2=%4.3f+-%4.3f  hist.sig=%4.3f+-%4.3f  cor.pairs=%ld'',
sd,sdd,ssig/ec,sssig/ec,lneq);
  if(mod==2) fprintf(fdat,
''\nim1-im2=%3.2f+-%3.2f  hist.sig=%2.1f+-%2.1f  cor.pairs=%ld'',
sd,sdd,ssig,sssig,lneq);
  fprintf(fdat,
''\nnum.cuts: im1 -%ld+%ld (%4.3f %4.3f)   im2 -%ld+%ld (%4.3f %4.3f)'',
  nco11,nco12,p11,p12,nco21,nco22,p21,p22);   fclose (fdat); }
```