

Igor recipes, Part I

A. Dauplain

January 14, 2010



The purpose of this document is to illustrate some Igor procedures useful for CFD posprocessing

Contents

1	The Fourier Transform	2
2	Custom color scales	3
3	A vector field over a matrix	5
4	Extracting numerical data from an image	7

1 The Fourier Transform

The Fourier Transform with Igor supposes first a good understanding of an 'Igor Wave': A wave is for Igor a serie of data (string, reals or integers) with a scaled dimension. For example, the serie Year defined as (Spring, Summer, Autumn, Winter) can be scaled by 1 to 365 , dimension 'day' (`SetScale x 0,365,"days",Year`), or by 1 to 12 , dimension 'month' (`SetScale x 1,12,"month",Year`). Refer to the help of 'Setscale' command for further details. Of course, Igor assumes the wave to be equally spaced. Varying timestep datasets must be interpolated first (see command 'Interp'). This preamble is necessary since *Igor compute spectrum only for datasets scaled in seconds 's'*. Using FFT on wrong scaled or not-scaled dataset leads to unpredictable results.

The following lines show the FFT of a simple synthetic signal:

```
// Signal, sinus 20Pa/120Hz
//      + sinus 15Pa/200Hz
//      + 5Pa white noise
fnyq=400
tmax=1/(2*fnyq)
Make/O/N=(sampling) wave0
SetScale/P x 0,tmax,"s",wave0
wave0= 20*cos(2*Pi*120*x) +15*cos(2*Pi*200*x) + enoise(5)

//FFT amplitude
sampling_zeros=(sampling*zeropad_factor)
FFT /PAD=(sampling_zeros)/OUT=3/DEST=fftwave/WINF=hanning wave0
fftwave=fftwave*4/sampling// to amplitude !
fftwave[0]=fftwave[0]/2// first freq exception
// WARNING, now the signal is the AMPLITUDE [Pa]
duplicate /O/C fftwave amplitude_by_fft

//Compute SPL in dB
duplicate /O/C fftwave SPLdB_by_fft
SPLdB_by_fft=SPLdB_by_fft*amptorms_factor // to RMS
SPLdB_by_fft=20*log(SPLdB_by_fft/20E-6) // to dB

//Compute DSP
duplicate /O/C amplitude_by_fft DSP_by_fft
DSP_by_fft=(DSP_by_fft)^2/4 // [Pa^2]
DSP_by_fft=DSP_by_fft*sampling// [Pa^2/Hz]
DSP_by_fft=DSP_by_fft
```

The 'zeropad' factor is used in the case of "peak hunting" to increase the number of points defining the graph of spectrum. More points will fall on the cardinal sinus function matching the discrete spectrum. Use values from 2 to 10. Zeropadding is forbidden in the case of DSP spectrums, use 1.

The 'amptorms' factor comes from the simple fact that the rms of a sine wave is equal to $\sqrt{2}/2$ times its amplitude. The definition of the Sound Pressure Level is based upon the rms pressure of the full signal. The procedure given here yields the SPL of each ray, independently from the other. The SPL

of the full signal cannot be computed through this FFT. Refer to the Cerfac technical report TR-CFD-10-8 "Four booby-traps in spectral analysis" for further informations.

2 Custom color scales

New colorscales are not obvious to define, but once you did one of them, they are easy to generalize. Three examples of custom color scale are given below:

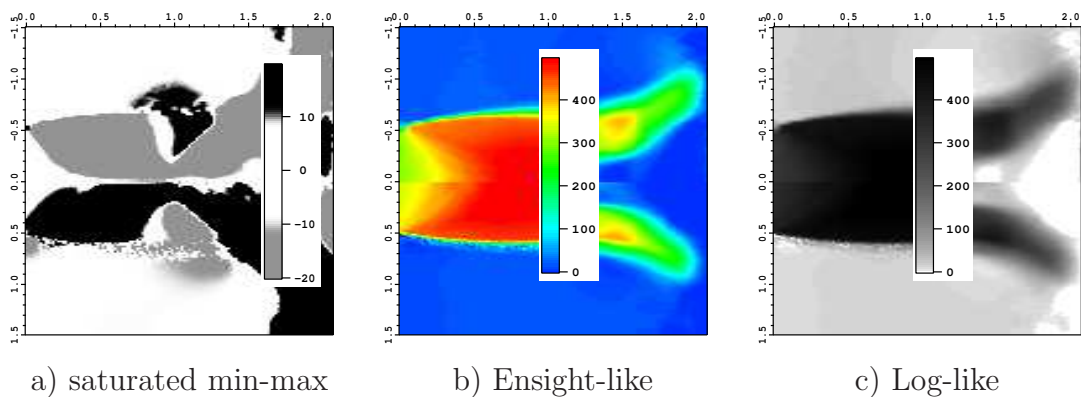


Figure 1: Three custom colorscales

1. The first is simply redefining the whole colors, with a small smooth at the end:

```
Function CreateMinMaxGrays()
Variable i, legsize, j
ColorTab2Wave Grays // Reads built-in color table into M_colors...
Duplicate/O M_colors, mycolors // ... which we rename
duplicate/O mycolors, mycolorsexp
legsize= DimSize(mycolors, 0)
for (i=0; i<=legsize-1; i+=1)
    mycolorsexp[i][0]=65535
    mycolorsexp[i][1]=65535
    mycolorsexp[i][2]=65535
    if (i< floor(legsize/4))
        mycolorsexp[i][0]=37767
        mycolorsexp[i][1]=37767
        mycolorsexp[i][2]=37767
    endif
    if (i> floor(3*legsize/4))
        mycolorsexp[i][0]=0
        mycolorsexp[i][1]=0
        mycolorsexp[i][2]=0
    endif
endfor
```

```
smooth /DIM=0 10,mycolorsexp
End
```

2. The second interpolate the initial rainbow colors scale on its first 85%, to get a more Ensign-looking palette:

```
Function CreateBlueGreenRed()
Variable i,legsize,j
ColorTab2Wave Rainbow // Reads built-in color table into M_colors...
Duplicate/O M_colors,mycolors // ... which we rename
duplicate/O mycolors, mycolorsexp
legsize= DimSize(mycolors, 0)
for (i=0;i<=legsize-1;i+=1)
    j= round((i/(legsize-1)) * 85 +15)
    j = legsize-j-1
    mycolorsexp[i][0]=mycolors[j][0]
    mycolorsexp[i][1]=mycolors[j][1]
    mycolorsexp[i][2]=mycolors[j][2]
endfor
End
```

3. The third make a Logscale-like scale, with a parameter alpha to control the bias of the scale:

```
Function CreateLogGrays(alpha)
Variable alpha
Variable i,legsize,j
ColorTab2Wave Grays // Reads built-in color table into M_colors...
Duplicate/O M_colors,mycolors // ... which we rename
duplicate/O mycolors, mycolorsexp
legsize= DimSize(mycolors, 0)
for (i=0;i<=legsize-1;i+=1)
    j= round((i/(legsize-1))^alpha * (legsize-1))
    j = legsize-j-1
    mycolorsexp[i][0]=mycolors[j][0]
    mycolorsexp[i][1]=mycolors[j][1]
    mycolorsexp[i][2]=mycolors[j][2]
endfor
setscale x,0,20,mycolorsexp
End
```

To enable these colors scales in ipf files:

```
CreateMinMaxGrays()
Duplicate /O mycolorsexp minmax
NewImage/K=0 root: phaselocked : V_comp_b
Setscale x, -20,20,minmax
ModifyImage V_comp_b cindex= minmax
ColorScale/C/N=text0/F=0/M/A=MC image=V_comp_b

CreateBlueGreenRed()
Duplicate /O mycolorsexp BGRlog
NewImage/K=0 root: phaselocked : U_comp_b
Setscale x,0,500,BGRlog
ModifyImage U_comp_b cindex= BGRlog
ColorScale/C/N=text0/F=0/M/A=MC image=U_comp_b
```

```

CreateLogGrays (0.5)
Duplicate /O mycolorsexp LogGray
NewImage/K=0 root : phaselocked : U_comp_b
Setscale x,0,500,LogGray
ModifyImage U_comp_b cindex= LogGray
ColorScale/C/N=text0/F=0/M/A=MC image=U_comp_b

```

Note the use of setscale to change the min and max of the scales...

3 A vector field over a matrix

Post-processing based upon a bundle of matrices is sometimes useful. In this domain, Igor is not Ensight or Tecplot but, "ad augusta, per angusta", nothing forbids you to develop your own dazzling graphical output for matrixes. The following function allows you to generate nice vector plots.

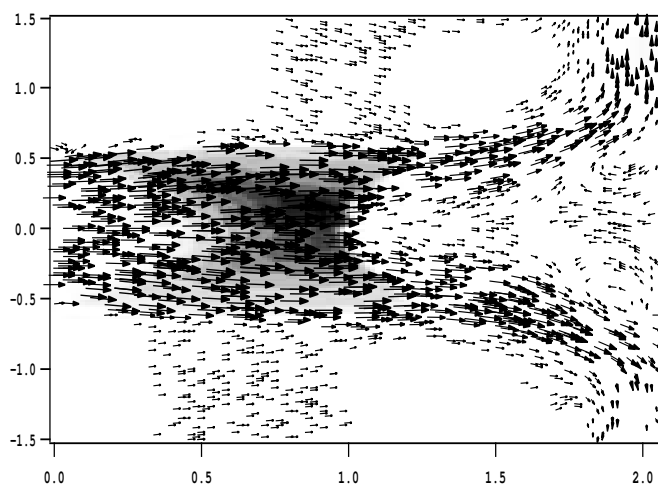


Figure 2: Vector plots by Igor

The function is easy to use:

```

VectorPlot (U_comp_a, V_comp_a, VEL_comp_MEAN, 0.0002, 0.1, 20)

```

This line was used for the creation of Fig. 2. The first and second matrix are the x and y component of the vectors. The third is the image to plot underneath (generally the magnitude, but you can be more creative).

Gain is the size of vectors. Find it with a trial-and-error process. Start with small values. Density is the number of vectors. Value 1 print all the pixels. 0.5 mean that half of the pixels will be skipped, randomly. Cutoff is

the level of matrixBACK under which you do not want vectors. It is useful to cancel the small vectors. Negative gain sets the constant-length mode (the value still controls the arrow size).

```
// *** Make a vector plot ***
Function VectorPlot(matrixU,matrixV,matrixBACK,gain,density,cutoff)
WAVE matrixU,matrixV,matrixBACK
Variable gain,density,cutoff
Variable i,j,k
Variable nx = DimSize(matrixU,0)
Variable ny=DimSize(matrixU,1)
Variable delta_x=DimDelta(matrixU,0)
Variable delta_y=DimDelta(matrixU,1)
Variable off_x=DimOffset(matrixU,0)
Variable off_y=DimOffset(matrixU,1)
Variable minx = off_x
Variable miny = off_y
Variable maxx= off_x + nx*delta_x
Variable maxy= off_y + ny*delta_y
// get samples on the matrix
Variable ns = floor(density*(nx*ny))
Variable step = floor(nx*ny/ns)
Print ns,step
Make/O/N=(ns) xw,yw,dx,dy
k=0
do
  i=floor(abs(enoise(nx)))
  j=floor(abs(enoise(ny)))
  xw[k]=off_x+(i)*delta_x
  yw[k]=off_y+(j)*delta_y
  dx[k]=matrixU[i][j]
  dy[k]=matrixV[i][j]
  if (sqrt(dx[k]^2+dy[k]^2) < cutoff)
    dx[k]=0
    dy[k]=0
  endif
  k += 1
while(k<ns)
CreateMinMaxGrays()
SetScale x,-1000,1000,mycolorsexp
Display;AppendImage matrixBACK
ModifyImage ''#0 ctab= {0,50,Rainbow,1}
ModifyImage ''#0 cindex= mycolorsexp
// original routine to make vects
Variable n=numpts(xw)
Variable minArrow= 1,maxArrow=8,minThick= 0.1, maxThick= 1
Variable minLen= 0.2
Duplicate/O dx, tmpALen
tmpALen= sqrt( (gain*dx)^2 + (gain*dy)^2 )
WaveStats/Q tmpALen
KillWaves tmpALen
Variable maxlen= V_max
SetDrawLayer/K ProgFront
SetDrawEnv xcoord= bottom,ycoord= left ,arrow= 1,arrowlen= 2.00,save
i=0
do
  Variable xo= xw[i],y0= yw[i], vdx= gain*dx[i], vdy= gain*dy[i]
  variable len= sqrt( vdx^2 + vdy^2 )
  Variable f=len/maxlen
  if (gain < 0) // constant size vector
    vdx= -gain * dx[i] / sqrt( dx[i]^2+ dy[i]^2)
```

```

vdy= -gain * dy[i] / sqrt( dx[i]^2+ dy[i]^2)
len= sqrt( vdx^2 + vdy^2 )
maxlen=1
f =0.2
endif

Variable thick= f*(maxThick-minThick) + minThick
//thick= 0.1
// vdx= f*vdx/2; vdy= f*vdy/2
vdx= vdx/f; vdy= vdy/f // normalized length
vdx= (vdx*minLen + vdx*((1-minLen)*f))/2
vdy= (vdy*minLen + vdy*((1-minLen)*f))/2
Variable ahead= f*(maxArrow-minArrow)+minArrow
SetDrawEnv linethick= thick ,arrowlen= ahead
DrawLine xo-vdx,y0-vdy,xo+vdx,y0+vdy
i += 1
while (i<n)
end

```

4 Extracting numerical data from an image

Sometimes, the data given in a communication is not accesible numerically. Oh! Frustration!... The following functions allow to extract the data from the image. In Fig. 3, the original data is given as a colored image, with

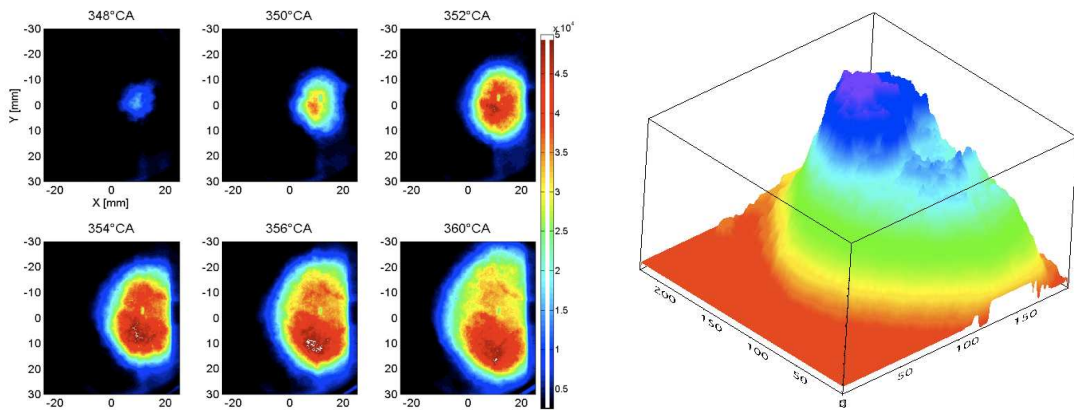


Figure 3: Image taken from an article, and data extracted from the last image

an unfriendly palette. The elevation graph on the left show the last field extracted via Igor.

First, the original images must be saved in the PPM ASCII format. It is compulsory to NOT include the borders of the field you want to extract. The color bar must also be saved in PPM format, using a selection one-pixel

wide (1x586 pixel for example). The Gimp software is free and perfect for this use. Then the PPM files must be edited and put into the IGOR text format. The PPM appear as:

```
P3 # type of PPM
# CREATOR: GIMP PNM Filter Version 1.1 # software creator
193 219 # size of the image
255 # nombre de niveaux de sampling (0 a 255)
0
0
...
```

Cancel the 4 first lines and write instead at the beginning and the end:

```
IGOR
WAVES image
BEGIN
0
...
0
END
```

And now, you can rush on your favourite ipf file :

This is the main macro. First, an image of 521x1 pixels is read as the legend (in color) Then the different images are read and transformed.

```
Macro GetLevels()
  SetDataFolder root:
  NewDataFolder /O ppm
  SetDataFolder root:ppm:
  Variable nlevel=20
  Variable levelmin = 0.3e+4
  Variable levelmax = 5e+4
  Variable ncols = 1 // pour une legende, il FAUT que cette image...
  ... fasse 1 seul pixel de large
  Variable nrows = 527
  Variable maxradius = 10
  String path = "DATA:GetLevels:"
  String name = "legend.igor"
  // les fichier .igor sont des PPM ascii.
  // On enleve les 4 premieres lignes et on ajoute au debut les 3 lignes :
  // IGOR
  // WAVES image
  // BEGIN
  // On ajoute a la fin les deux lignes
  // END
  //
  Duplicate/O ReadLegendPPM(path,name,ncols,nrows,levelmin,...
  ... levelmax,nlevel,maxradius), level_gobal
  NewImage/K=0 root:ppm:level_gobal
  ModifyImage level_gobal ctab= {1,256,Rainbow,1}

  //Read the source image
  ncols = 195 // size of the ppm image
  nrows = 220
  Variable tolerance = 40 // maximum distance to color levels (4x min radius)
  name = "benoit_348_195x220.ppm"
  Duplicate/O ReadPPM(path,name,ncols,nrows,level_gobal,nlevel,...
  ... tolerance,0), XP_348
  NewImage/K=0 root:ppm:XP_348
```

```

ModifyImage XP_348 ctab= {1,*,Rainbow,1}
ColorScale/C/N=text0/M/A=LC image=XP_348

//Read the source image
ncols = 194 // size of the ppm image
nrows = 220
tolerance = 40 // maximum distance to color levels (4x min radius)
name = "benoit_350_194x220.ppm"
Duplicate/O ReadPPM(path,name,ncols,nrows,level_gobal,nlevel,...
...tolerance,0), XP_350
NewImage/K=0 root:ppm:XP_350
ModifyImage XP_350 ctab= {1,*,Rainbow,1}
ColorScale/C/N=text0/M/A=LC image=XP_350

//Read the source image
ncols = 194 // size of the ppm image
nrows = 220
tolerance = 40 // maximum distance to color levels (4x min radius)
name = "benoit_352_194x220.ppm"
Duplicate/O ReadPPM(path,name,ncols,nrows,level_gobal,nlevel,...
...tolerance,0), XP_352
NewImage/K=0 root:ppm:XP_352
ModifyImage XP_352 ctab= {1,*,Rainbow,1}
ColorScale/C/N=text0/M/A=LC image=XP_352

//Read the source image
ncols = 195 // size of the ppm image
nrows = 220
tolerance = 40 // maximum distance to color levels (4x min radius)
name = "benoit_354_195x220.ppm"
Duplicate/O ReadPPM(path,name,ncols,nrows,level_gobal,nlevel,...
...tolerance,0), XP_354
NewImage/K=0 root:ppm:XP_354
ModifyImage XP_354 ctab= {1,*,Rainbow,1}
ColorScale/C/N=text0/M/A=LC image=XP_354

//Read the source image
ncols = 195 // size of the ppm image
nrows = 219
tolerance = 40 // maximum distance to color levels (4x min radius)
name = "benoit_356_195x219.ppm"
Duplicate/O ReadPPM(path,name,ncols,nrows,level_gobal,nlevel...
...,tolerance,0), XP_356
NewImage/K=0 root:ppm:XP_356
ModifyImage XP_356 ctab= {1,*,Rainbow,1}
ColorScale/C/N=text0/M/A=LC image=XP_356

//Read the source image
ncols = 195 // size of the ppm image
nrows = 220
tolerance = 40 // maximum distance to color levels (4x min radius)
name = "benoit_360_195x220.ppm"
Duplicate/O ReadPPM(path,name,ncols,nrows,level_gobal,nlevel...
...,tolerance,0), XP_360
NewImage/K=0 root:ppm:XP_360
ModifyImage XP_360 ctab= {1,*,Rainbow,1}
ColorScale/C/N=text0/M/A=LC image=XP_360

//Interpolation
//Variable ncols_out = 200 // Dimension de la grille ...
...d'interpolation (200x200 est deja pas mal)
//Variable nrows_out = 200

```

```

//Variable nsamples = 4000 // nombre de points a ...
... ,distribuer sur les contours (1000<nsamples<10000)
//Duplicate/O getXYZcontours(ncols ,nrows ,XP_349 ,ncols_out ,...
... nrows_out ,nsamples), XP_smooth
//NewImage/K=0 root:ppm:XP_smooth
//ModifyImage XP_smooth ctab= {1,*,Rainbow,1}
//ColorScale/C/N=text0/M/A=LC image=XP_smooth
// KillWaves /A/Z
SetDataFolder root:
EndMacro

```

Note that in the last commented section, one of the matrixes is resampled, then interpolated on a smaller matrix. This is done to get a continuous variation of levels. After this process, you will be able to cut profiles in your datasets which are smooth, setting you free of the aliasing of the 20 (nlevel value equal 20 in this example) colors available on the color scale. Here are the functions caled within the routine:

First, a function to close all graphs (a classic):

```

Function CloseAllGraphs ()
String name
do
    name = WinName(0,1) // name of the front graph
    if (strlen(name) == 0)
        break // all done
    endif
    DoWindow/K $name // Close the graph
while(1)
End

```

This function finds automatically 'nlevel' in the color scale. This 'nlevel' colors will be the reference palette :

```

// *** The reading of PPM images legend ***
Function/wave ReadLegendPPM(path ,name ,columns ,rows ,levelmin ,levelmax ,...
... nlevel ,maxradius)
Variable columns ,rows , levelmin ,levelmax ,nlevel ,maxradius
String path ,name

// local variables
Variable ilevel ,value
Variable i ,j ,r ,g ,b ,radius
Variable last_level = -10
Variable ro = 1000
Variable go = 1000
Variable bo = 1000
Variable min_radius = 100000

print ">>>>>>_Begin_legend_Analysis..."
// read file into imageRGB
Newpath /O path_local , path
loadwave /O /T/P=path_local , name
String data = "image"
duplicate /O $data imageRGB
imageRGB = imageRGB +1

// create intermediate levels
MAKE /O/N=(4,nlevel) level

```

```

for (i=0;i<=rows-1;i+=1)
for (j=0;j<=columns-1;j+=1)
g=imageRGB[ ((i)*columns+j+1)*3-2]
b=imageRGB[ ((i)*columns+j+1)*3-1]
r=imageRGB[ ((i)*columns+j+1)*3]

ilevel = round((i+1)/rows * nlevel)

value = levelmax - (levelmax-levelmin)*ilevel/nlevel

level[][ilevel]={r,g,b,value}

// tst on color distinction
radius = sqrt((r-ro)^2+(g-go)^2+(b-bo)^2)
min_radius = min(min_radius,radius)

if (ilevel-last_level > 0 )
last_level=ilevel
radius = sqrt((r-ro)^2+(g-go)^2+(b-bo)^2)
min_radius = min(min_radius,radius)
ro=r
bo=g
go=g
endif
endfor
endfor

print "End_of_Reading..."
print "Minimal_color_distinction_", min_radius, "(Above_5_is_good)"
return level
End

```

This function convert the PPM image into a matrix, Z being sampled upon the previous color scale. On the matrix, Z values will be one of the 'nlevels' colors sampled previously, or unassigned (value -2). Unassigned means that the color on this pixel was too far from the 'nlevels' samples in the RGB coordinates. Assign the missing value is done to propagate the colors known, filling the unassigned regions:

```

// *** The reading of PPM images ***
Function/wave ReadPPM(path,name,columns,rows,level,nlevel,maxradius,...
... smoothstencil)
Variable rows, columns, nlevel, maxradius, smoothstencil
Wave level
String path, name
Variable i, j, r, g, b, k, ncolors, found, radius, last_radius
Variable ii, jj, avg, avgc
Variable progress, last_time=0
Newpath /O path_local, path
loadwave /O /T/P=path_local, name
String data = "image"
duplicate /O $data imageRGB
imageRGB = imageRGB +1
MAKE /O/N=(columns,rows) Red
MAKE /O/N=(columns,rows) Green
MAKE /O/N=(columns,rows) Blue
MAKE /O/N=(columns,rows) Red_c

```

```

MAKE /O/N=(columns , rows) Green_c
MAKE /O/N=(columns , rows) Blue_c
MAKE /O/N=(columns , rows) Result
MAKE /O/N=(columns , rows) Res_radius = -1
MAKE /O/N=(columns , rows) Mask = 1

print ">>>>>>_Begin_image_analysis..."

for (i=0;i<=rows-1;i+=1)
for (j=0;j<=columns-1;j+=1)
  progress =((i)*columns +j+1)/(rows*columns)*100
  if (progress-last_time>10)
    last_time =progress
    print progress,"%"
  endif

  Green[j][i]=imageRGB[ ((i)*columns +j+1)*3-2]
  Blue[j][i]=imageRGB[ ((i)*columns +j+1)*3-1]
  Red[j][i]=imageRGB[ ((i)*columns +j+1)*3]
  r=Red[j][i]
  g=Green[j][i]
  b=Blue[j][i]

  // identification de la couleur
  last_radius = 1000000
  for (k=1;k<=nlevel;k+=1)
    radius = sqrt((r-level[0][k])^2+(g-level[1][k])^2+(b-level[2][k])^2)
    if (radius < last_radius)
      Red_c[j][i] = level[0][k]
      Green_c[j][i] = level[1][k]
      Blue_c[j][i] = level[2][k]
      Result[j][i]= level[3][k]
      last_radius=radius
      Res_radius[j][i]= radius
    endif
  endfor

  if (last_radius > maxradius)
    Result[j][i]=-2
  endif

  // Identification mask
  if ( Result[j][i]<=-2 )
    Mask[j][i]=0
  endif

endfor
endfor

print "End of Reading..." , progress,"%"

// Images
//NewImage/K=0 root:ppm:Mask
//NewImage/K=0 root:ppm:res_radius
//ModifyImage res_radius ctab= {0,maxradius,Terrain256,1}
//ColorScale/C/N=text0/A=MC image=Res_radius

//NewImage/K=0 root:ppm:mask
//ModifyImage mask ctab= {*,*,Grays,1}

//NewImage/K=0 root:ppm:result
//ModifyImage Result ctab= {1,*,Rainbow,1}

```

```

//ModifyImage Result minRGB=(0,0,0),maxRGB=0

Duplicate /O Mask NewMask
// assign the missing values
for (k=1;k<=5;k+=1)
  Mask = NewMask
  for (i=1;i<=rows-1;i+=1)
    for (j=1;j<=columns-1;j+=1)
      If (Mask[j][i]==0)
        avg = 0
        avgc = 0
        for (ii=-1;ii<=1;ii+=1)
          for (jj=-1;jj<=1;jj+=1)
            if (Mask[j+ii][i+ii]==1)
              avg = avg +Result[j+ii][i+ii]
              avgc +=1
            endif
          endfor
        endfor
      endif
      if (avgc >=1)
        Result[j][i] =(avg / avgc)
      endif
      if (avgc >=3)
        NewMask[j][i] = 1
      endif
    endif
  endfor
endfor
endfor

// smooth
matrixfilter /N=(smoothstencil) gauss Result
return Result
End

```

This one is not easy : FindEdges finds the location of the frontier between two levels. These locations are sampled randomly, creating triplets XYZ. The edges are more sampled, to get a better resolution. Good news, this function is needed only if you want to get smooth datasets, to cut a profile inside for example.

```

// get triples from contours
Function /wave getXYZcontours (columns , rows , wave1 , columns_out , rows_out , ...
... nsample_max)
Variable rows , columns
Variable rows_out , columns_out , nsample_max
Wave wave1
KillWaves /Z wave_xyz
Variable i , j , nsample , ii , jj , res , ii_index , jj_index
Make/O/N=(nsample_max , 3) wave_xyz
print ">>>>>>_Begin_image_interpolation ... "
// create edges
duplicate /O wave1 edegesmtx
MatrixFilter FindEdges edegesmtx
Duplicate /O edegesmtx sammtx
sammtx=0
nsample =4
wave_xyz [0][0]=0
wave_xyz [0][1]=0
wave_xyz [0][2]=wave1 [0][0]

```

```

wave_xyz [1][0]=1
wave_xyz [1][1]=0
wave_xyz [1][2]=wave1 [ columns -1][0]
wave_xyz [2][0]=0
wave_xyz [2][1]=1
wave_xyz [2][2]=wave1 [0][ rows -1]
wave_xyz [3][0]=1
wave_xyz [3][1]=1
wave_xyz [3][2]=wave1 [ columns -1][ rows -1]
// borders
for ( i=0;i<=rows-1;i+=1)
    j=0
    if (100*(enoise (0.5,2) + 0.5)<=10)
        nsample =min( nsample_max , nsample +1)
        sammtx [ j ][ i ] = 1
        wave_xyz [ nsample ][0]= j /( columns -1)
        wave_xyz [ nsample ][1]= i /( rows -1)
        wave_xyz [ nsample ][2]= wave1 [ j ][ i ]
    endif
    j=columns-1
    if (100*(enoise (0.5,2) + 0.5)<=10)
        nsample =min( nsample_max , nsample +1)
        sammtx [ j ][ i ] = 1
        wave_xyz [ nsample ][0]= j /( columns -1)
        wave_xyz [ nsample ][1]= i /( rows -1)
        wave_xyz [ nsample ][2]= wave1 [ j ][ i ]
    endif
endfor
for ( j=0;j<=columns-1;j+=1)
    i=0
    if (100*(enoise (0.5,2) + 0.5)<=10)
        nsample =min( nsample_max , nsample +1)
        sammtx [ j ][ i ] = 1
        wave_xyz [ nsample ][0]= j /( columns -1)
        wave_xyz [ nsample ][1]= i /( rows -1)
        wave_xyz [ nsample ][2]= wave1 [ j ][ i ]
    endif
    i=rows-1
    if (100*(enoise (0.5,2) + 0.5)<=10)
        nsample =min( nsample_max , nsample +1)
        sammtx [ j ][ i ] = 1
        wave_xyz [ nsample ][0]= j /( columns -1)
        wave_xyz [ nsample ][1]= i /( rows -1)
        wave_xyz [ nsample ][2]= wave1 [ j ][ i ]
    endif
endfor

for ( i=0;i<=rows-1;i+=1)
    for ( j=0;j<=columns-1;j+=1)
        if (edgesmtx [ j ][ i ] >=50)
            if ((100*(enoise (0.5,2) + 0.5)<=10) )//|| ( i <=1)|| ( i >=rows-2) ...
                ... || ( j <=1) || ( j >= columns -2))
                sammtx [ j ][ i ] = 1
                wave_xyz [ nsample ][0]= j /( columns -1)
                wave_xyz [ nsample ][1]= i /( rows -1)
                res = wave1 [ j ][ i ]
                for ( ii=-2;ii<=2;ii+=1)
                    for ( jj=-2;jj <=2;jj+=1)
                        ii_index = max(0, i+ii)
                        ii_index = min( rows -1, i+ii)
                        jj_index = max(0, j+jj)
                        jj_index = min( columns -1, j+jj)

```

```

        if (wave1[jj_index][ii_index]<res)
            res=wave1[jj_index][ii_index]
        endif
    endfor
endfor
wave_xyz[nsample][2]=res
if (nsample == nsample_max)
    sammtx[j][i] = 0
endif
nsample =min(nsample_max , nsample +1)
endif
endif
endfor
endfor
print "Samples found ", nsample, "max samples auth are:", nsample_max
duplicate /O/R=(0,nsample) wave_xyz wave_xyz_trim
imageinterpolate /S={0,1/(columns_out-1),1,0,1/(rows_out-1),1}/E=0 ...
.../DEST=matrix_interp voronoi wave_xyz_trim
matrixfilter /N=(3) gauss matrix_interp
setscale x 0,1,matrix_interp
setscale y 0,1,matrix_interp
Return matrix_interp
End

```

One of the most interesting features in this function is the ability of creating a continuous (non aliased) matrix from the contour. To illustrate the process, this short function , not used in the Getlevels macro, shows how to go from a 2D could of XYZ points into a matrix

```

// transforme une onde XYZ en matrice nxm
Function /wave InterpXYZ()
Variable n,m
//KillWaves wave1
Variable nsamples = 30
Variable i,x,y,z
MAKE /O/N=(nsamples,3) wave1
for (i=0;i<=nsamples-1;i+=1)
x = (enoise (0.5,2) + 0.5)*2*PI
y = (enoise (0.5,2) + 0.5)*2*PI
z = cos(x)*sin(y)
wave1[i][0]=x
wave1[i][1]=y
wave1[i][2]=z
endifor
imageinterpolate /S={0,PI/20,2*PI,0,PI/20,2*PI} voronoi wave1
End

```