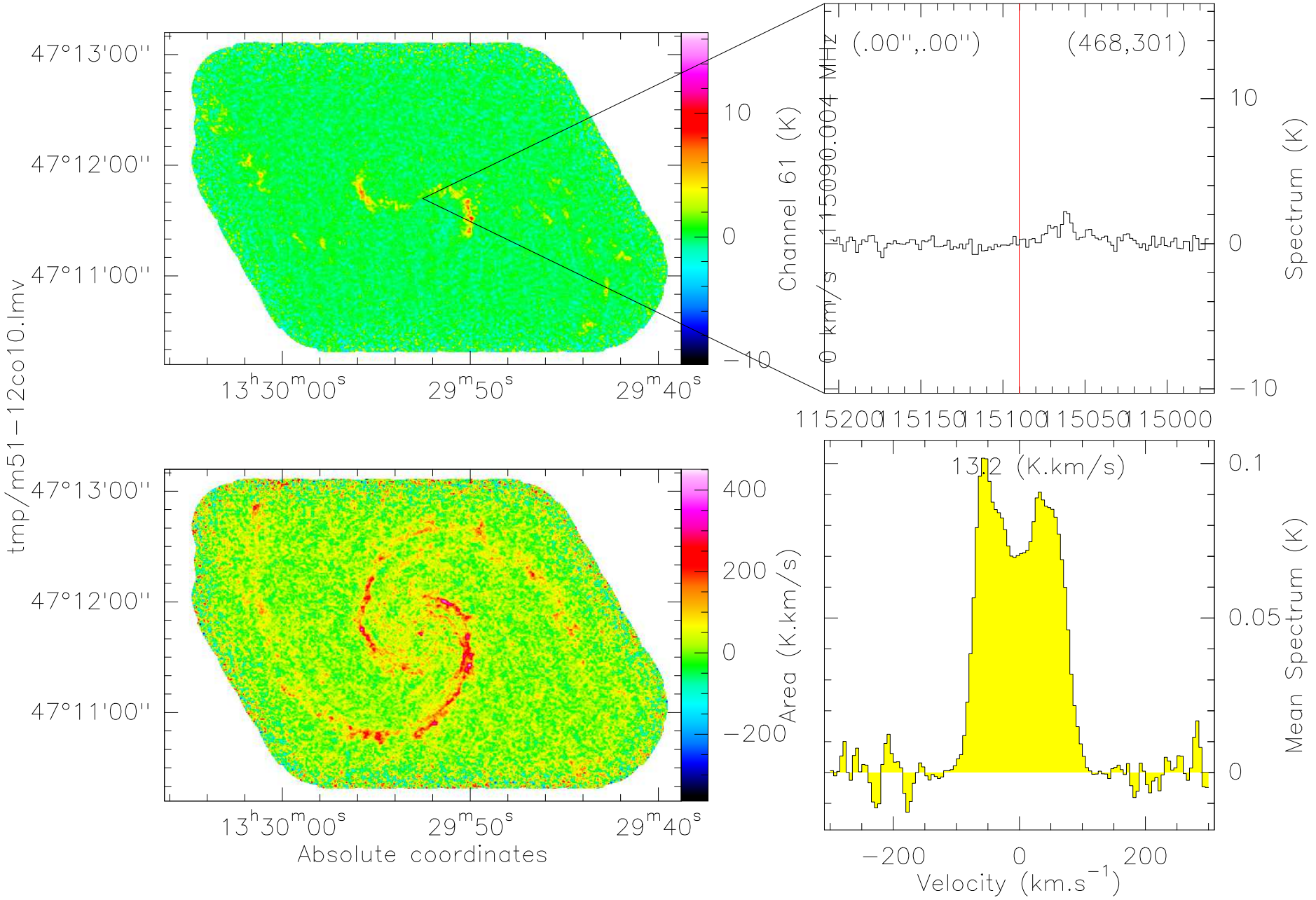
GREG tutorial: III. Building complex figures

Presentation by Jérôme PETY & Sébastien BARDEAU
(IRAM/Grenoble)
Current kernel developers: Jérôme PETY, Sébastien BARDEAU, &
Stéphane GUILLOTEAU
on behalf of the GILDAS developers over time

July 2019, IRAM/Granada

Going from already coded tools (e.g., GO VIEW)



To user-customed publication-quality figure

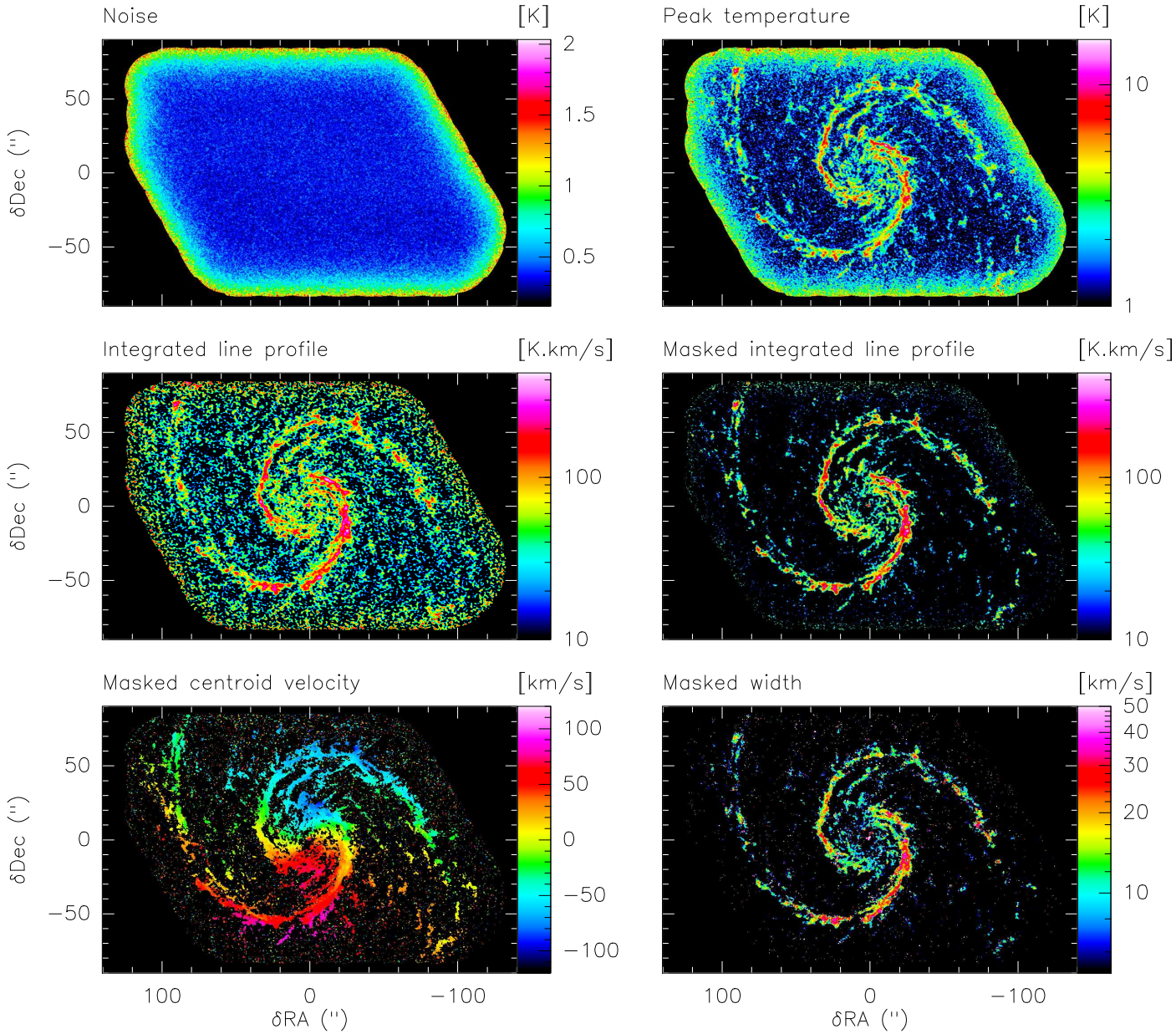
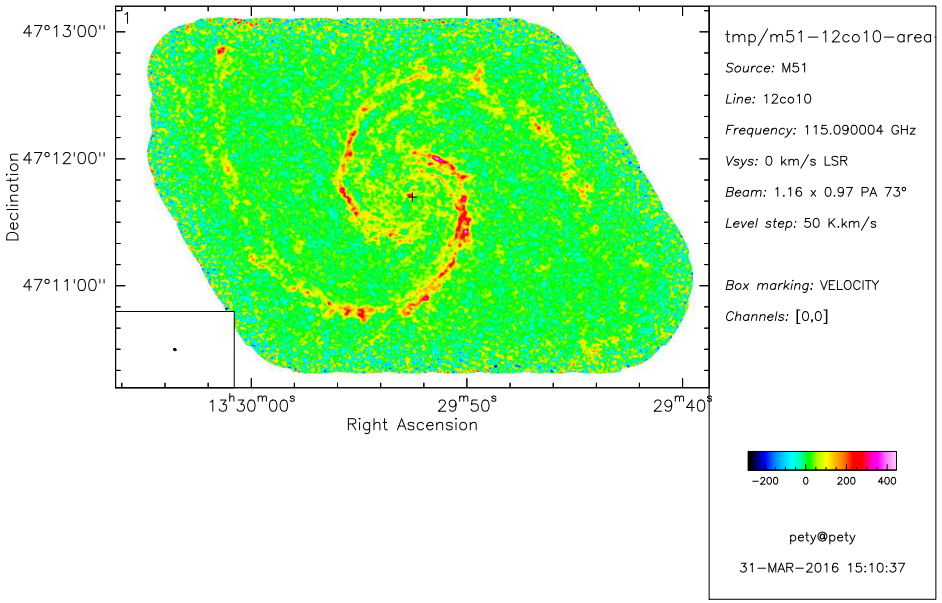


Table of contents

Computing "moments": I. Line integrated profile

Procedure moments-area in file pro/moments.greg

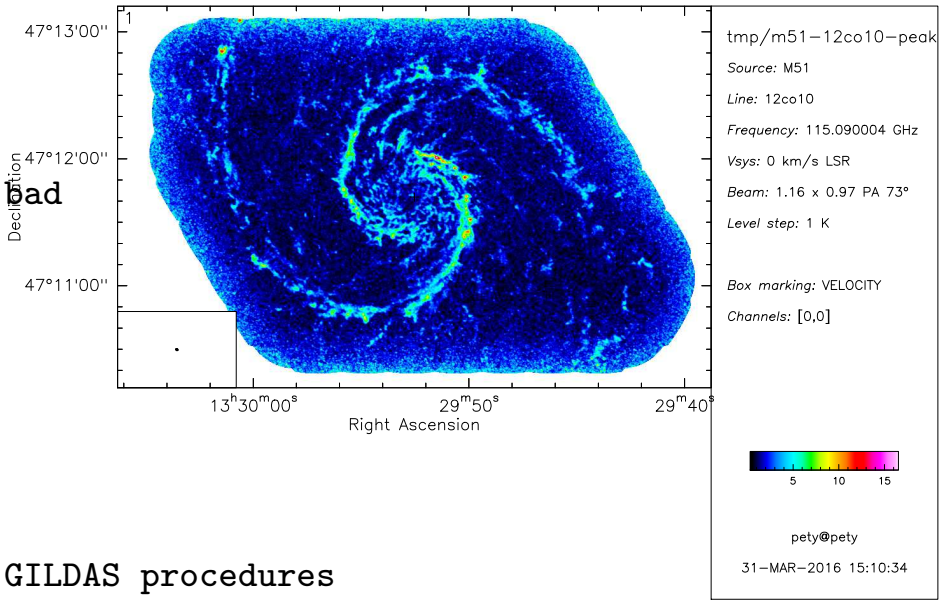
```
begin procedure moments-area
  let name tmp/m51-12co10
  let type lmv
  ! Integrate brightness temperatures
  ! between channels 31 and 90
  let first 31
  let last  90
  go area ! The tool already exists => Easy
  ! Name set to tmp/m51-12co10-area
  ! Reset first and last to avoid plotting
  ! the same image 60 times
  let first 0
  let last 0
  go nice
end procedure moments-area
!
let do_contour no     ! No contours on M51 => Better images
@ moments-area        ! All image related definitions
```



Computing "moments": II. Peak temperature

Procedure moments-peak in file pro/moments.greg

```
begin procedure moments-peak
  let name tmp/m51-12co10
  let type lmv
  ! Search for the spectrum maximum
  ! between channels 31 and 90
  let first 31
  let last  90
  @ cube-peak ! This ones does not exist => Too bad
  ! Name set to tmp/m51-12co10-peak
  ! Reset first and last to avoid plotting
  ! the same image 60 times
  let first 0
  let last 0
  go nice
end procedure moments-peak
!
! Add the ./tools directory on the path to find GILDAS procedures
sic log macro#dir: "./;./pro/;./tools/;"'macro#dir:'
@ cube-tools          ! Load the LMV cube related tools
@ moments-peak        ! All image related definitions
```



Computing "moments": II. Peak temperature

1. Loading the data cube, checking this is a LMV cube, computing the channel range from the input variables

Procedure cube-load in file tools/cube-tools.greg

```
! Check the SIC tutorial first ;-)
@ cube-load  aaa1
```

Procedure cube-check in file tools/cube-tools.greg

```
@ cube-check  aaa1
!
begin procedure cube-check
  if ((&1%x_axis.ne.1).or.(&1%y_axis.ne.2).or.(&1%f_axis.ne.3)) then
     message e cube-check "Not an LMV cube => Please transpose"
     return base
  endif
end procedure cube-check
```

Procedure cube-range in file tools/cube-tools.greg

```
@ cube-range  aaa1
!
begin procedure cube-range
  if (first.le.0) then
     let cube%first 1
  else
     let cube%first first
  endif
  if (last.le.0) then
     let cube%last &1%dim[&1%f_axis]
  else
     let cube%last last
  endif
end procedure cube-range
```

Computing "moments": II. Peak temperature

2. User feedback

Procedure cube-tpeak in file tools/cube-tools.greg

```
@ cube-load  aaa1
@ cube-check aaa1
@ cube-range aaa1
message i cube-tpeak "Channel range used ["'cube%first'":"'cube%last'"]"
```

Computing "moments": II. Peak temperature

3. Opening the output file, updating the header and the data, updating the extrema

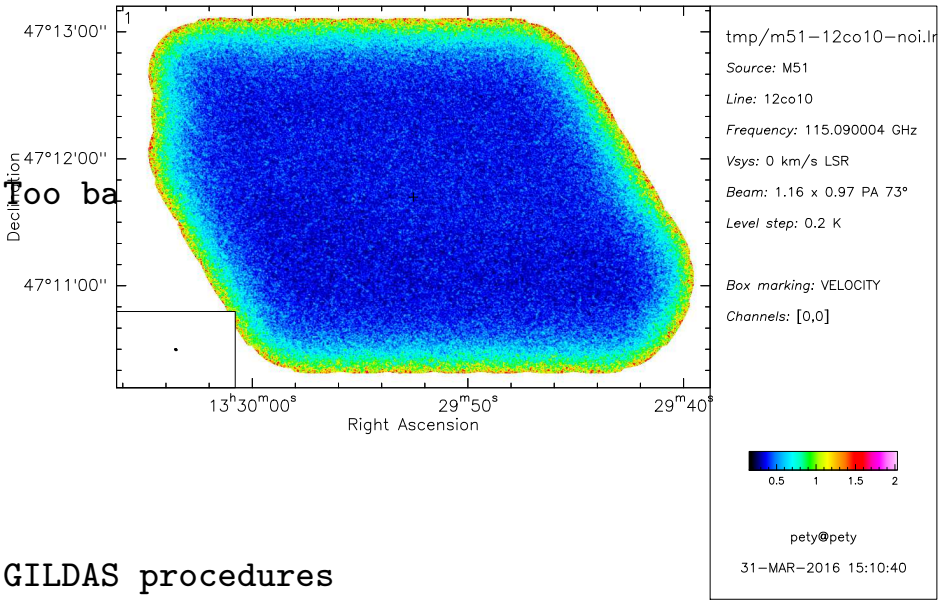
Procedure cube-tpeak in file tools/cube-tools.greg

```
define image tpeak 'name'"-peak."'type' real /like aaa1[1]
!
let tpeak% aaa1% Copying the header of the input LMV cube
let tpeak%convert[3] 0 The output is an image => No more frequency axis!
let tpeak%f_axis 0     The output is an image => No more frequency axis!
! *** Extremely slow code (non-contiguous memory access + non-vectorized loop) ***
!  for ix 1 to aaa1%dim[1]
!     if (mod(ix,nint(aaa1%dim[1]/10)).eq.0) then
!        message r cube-tpeak "Done "'nint(100*ix/aaa1%dim[1])'" %"
!     endif
!     for iy 1 to aaa1%dim[2]
!        compute tpeak[ix,iy] max aaa1[ix,iy,'cube%first':'cube%last'] /blanking aaa1%blank[1] aaa1%blank
!     next iy
!  next ix
! *** Extremely slow code (non-contiguous memory access + non-vectorized loop) ***
! *** Fast code ***
compute tpeak max aaa1['cube%first':'cube%last'] /blanking aaa1%blank[1] aaa1%blank[2]
! *** Fast code ***
! Typical peak brightness as user feedback
define double med
compute med median tpeak /blanking tpeak%blank[1] tpeak%blank[2]
message i cube-tpeak "Median peak brigthness "'med'" "'tpeak%unit'
delete /variable tpeak ! That's where the output is written on disk
! Redefine the NAME variable for next action
let name 'name'"-peak"
message w cube-tpeak "Name set to "'name'
! Update extrema
header 'name'"."'type' /extrema
```

Computing "moments": III. Noise spatial distribution

Procedure moments-noise in file pro/moments.greg

```
begin procedure moments-noise
  let name tmp/m51-12co10
  let type lmv
  ! Compute rms of line-free channels,
  ! i.e., between channels 1 and 30
  let first  1
  let last  30
  @ noise-compute ! This ones does not exist => Too ba
  ! Name set to tmp/m51-12co10-noise
  ! Reset first and last to avoid plotting
  ! the same image 30 times
  let first 0
  let last 0
  go nice
end procedure moments-noise
!
! Add the ./tools directory on the path to find GILDAS procedures
sic log macro#dir: "./;./pro/;./tools/;"'macro#dir:'
@ noise-tools         ! Load the LMV cube related tools
@ moments-noise       ! All image related definitions
```



Computing "moments": III. Noise spatial distribution

1. Loading the data cube, checking this is a LMV cube, computing the channel range from the input variables, user feedback

Procedure noise-compute in file tools/noise-tools.greg

```
! Exactly the same start as the computation of the line peak ;-)
@ cube-load  aaa1
@ cube-check aaa1
@ cube-range aaa1
message i cube-tpeak "Channel range used ["'cube%first'":"'cube%last'"]"
```

Computing "moments": III. Noise spatial distribution

2. Opening the output file, updating the header and the data, updating the extrema

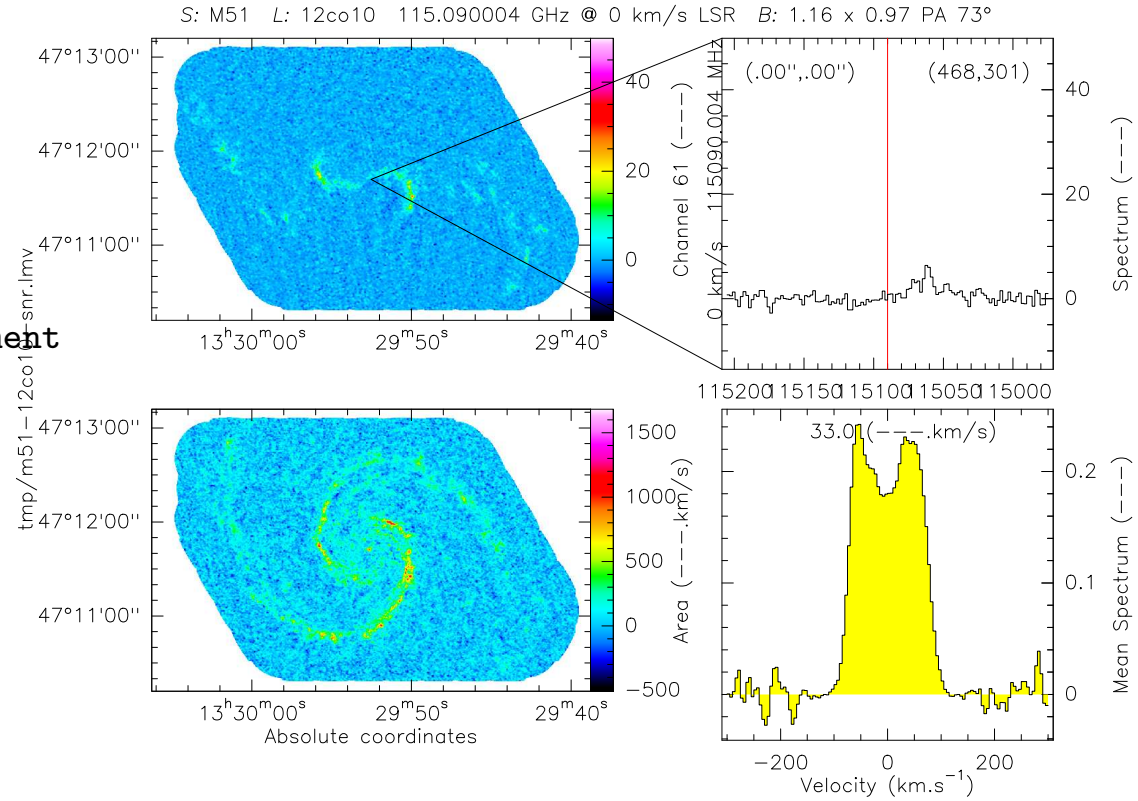
Procedure noise-compute in file tools/noise-tools.greg

```
! An almost verbatim copy of the cube-tpeak procedure ;-)
define image noise 'name'"-noi."'type' real /like aaa1[1]
!
let noise% aaa1% Copying the header of the input LMV cube
let noise%convert[3] 0 The output is an image => No more frequency axis!
let noise%f_axis 0     The output is an image => No more frequency axis!
compute noise rms aaa1['cube%first':'cube%last'] /blanking aaa1%blank[1] aaa1%blank[2]
! Typical peak brightness as user feedback
define double med
compute med median noise /blanking noise%blank[1] noise%blank[2]
message i noise-compute "Median rms "'med'" "'noise%unit'
delete /variable noise ! That's where the output is written on disk
! Redefine the NAME variable for next action
let name 'name'"-noi"
message w noise-compute "Name set to "'name'
! Update extrema
header 'name'"."'type' /extrema
```

Computing "moments": IV. Signal-to-Noise Ratio

Procedure moments-snr in file pro/moments.greg

```
begin procedure moments-snr
  let name tmp/m51-12co10
  let type lmv
  ! Interpret the NAME variable and
  ! pass it as the procedure 1st argument
  @ noise-snr 'name'
  ! Name set to tmp/m51-12co10-snr
  go view
end procedure moments-snr
!
@ noise-tools
@ moments-snr
```



Computing "moments": IV. Signal-to-Noise Ratio

1. Checking the input parameters, loading the data cube and the noise image, and opening the output file

Procedure noise-snr in file tools/noise-tools.greg

```
if (pro%narg.ne.1) then
   message e noise-snr "Usage: @ noise-snr name"
   return base
endif
let name "&1"
@ cube-load sig
let name "&1-noi"
@ cube-load noi
define image snr "&1-snr."'type' real /like sig
```

Computing "moments": IV. Signal-to-Noise Ratio

2. Update the header and the data, update the extrema

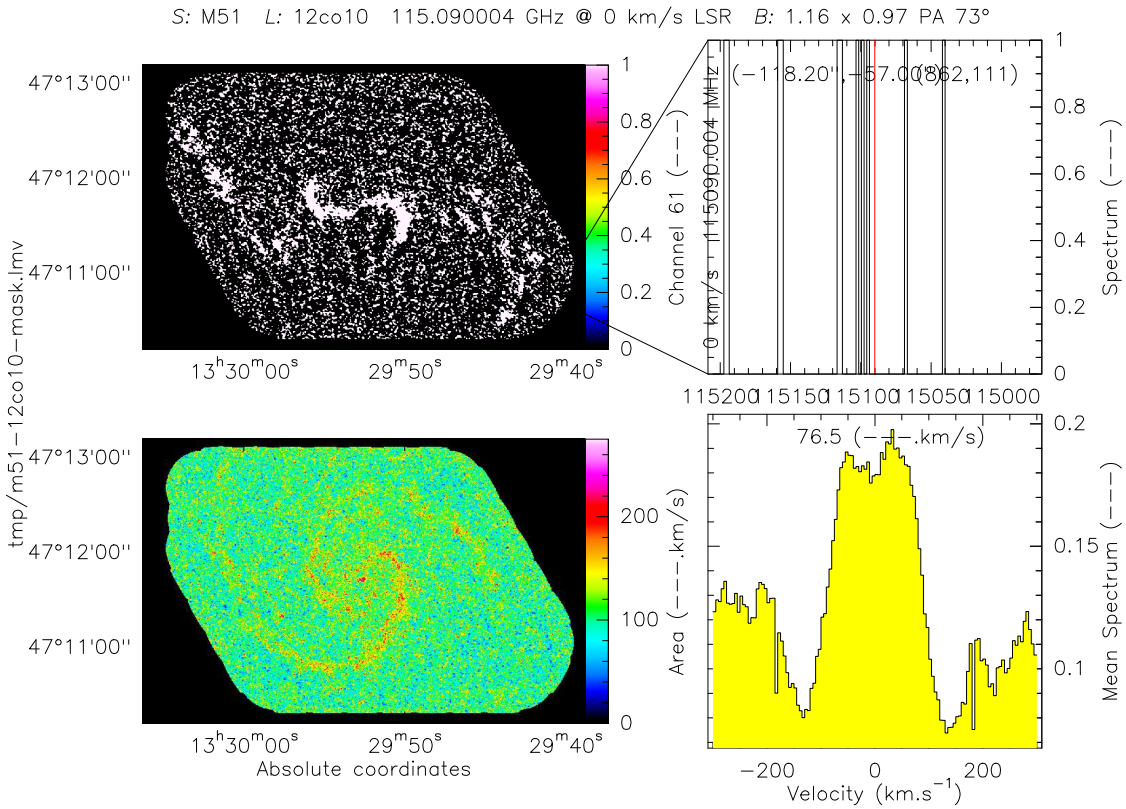
Procedure noise-snr in file tools/noise-tools.greg

```
let snr% sig%          ! Copy the header
let snr%unit "---"     ! No unit for a SNR
let snr snr%blank[1] ! Initialize to the blanking value
! Loop over the channels
for ichan 1 to snr%dim[3]
   ! The following command is vectorized over the ix and iy axes
   ! The computation is done only where the pixel value is
   ! different from the blank value
   let snr[ichan] sig[ichan]/noi -
       /where (abs(sig[ichan]-sig%blank[1]).gt.sig%blank[2]).and.(abs(noi-noi%blank[1]).gt.noi%blank[2])
next ichan
delete /var snr noi sig
let name "&1-snr"
message w noise-snr "Name set to "'name'
header 'name'"."'type' /extrema
```

Computing "moments": V. Finding noisy pixels

Procedure moments-mask-define in file pro/moments.greg

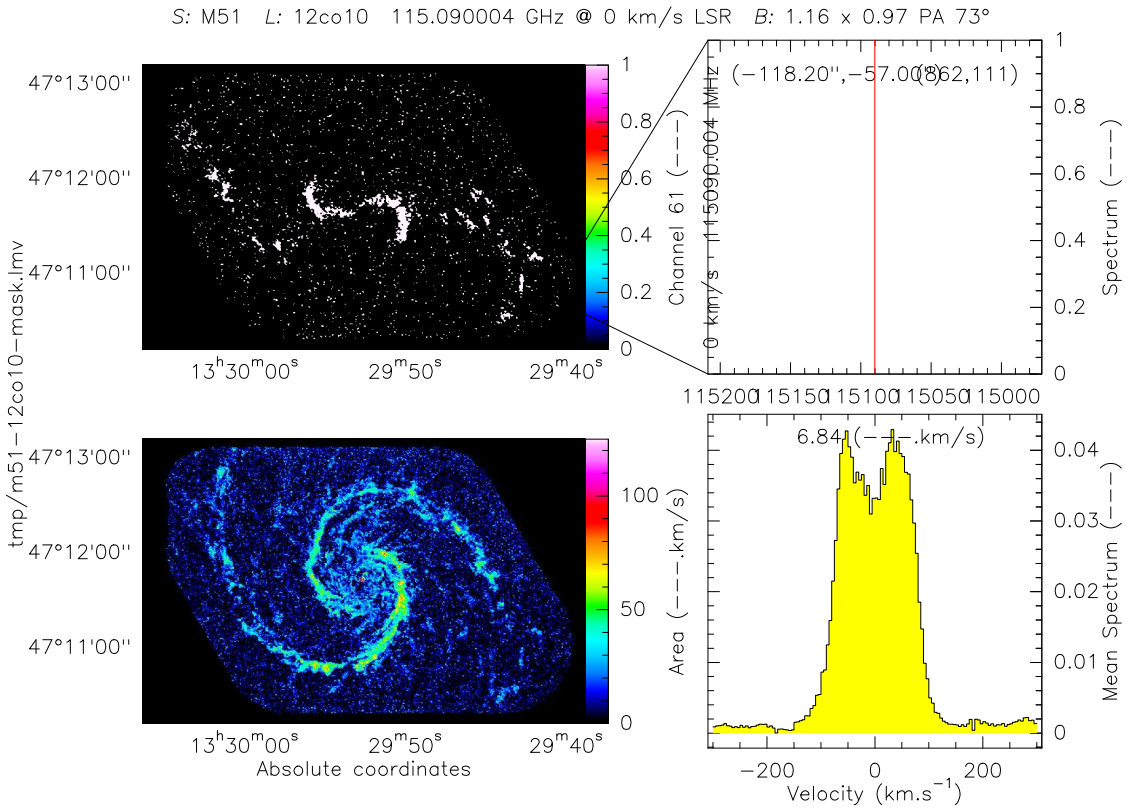
```
begin procedure moments-mask-define
  let name tmp/m51-12co10
  let type lmv
  ! Pass the SNR threshold used
  ! define the mask
  @ noise-mask-define 1
  ! Name set to tmp/m51-12co10-mask
  go view
end procedure moments-mask-define
!
@ noise-tools
@ moments-mask-define
```



Computing "moments": V. Finding noisy pixels

Procedure moments-mask-define in file pro/moments.greg

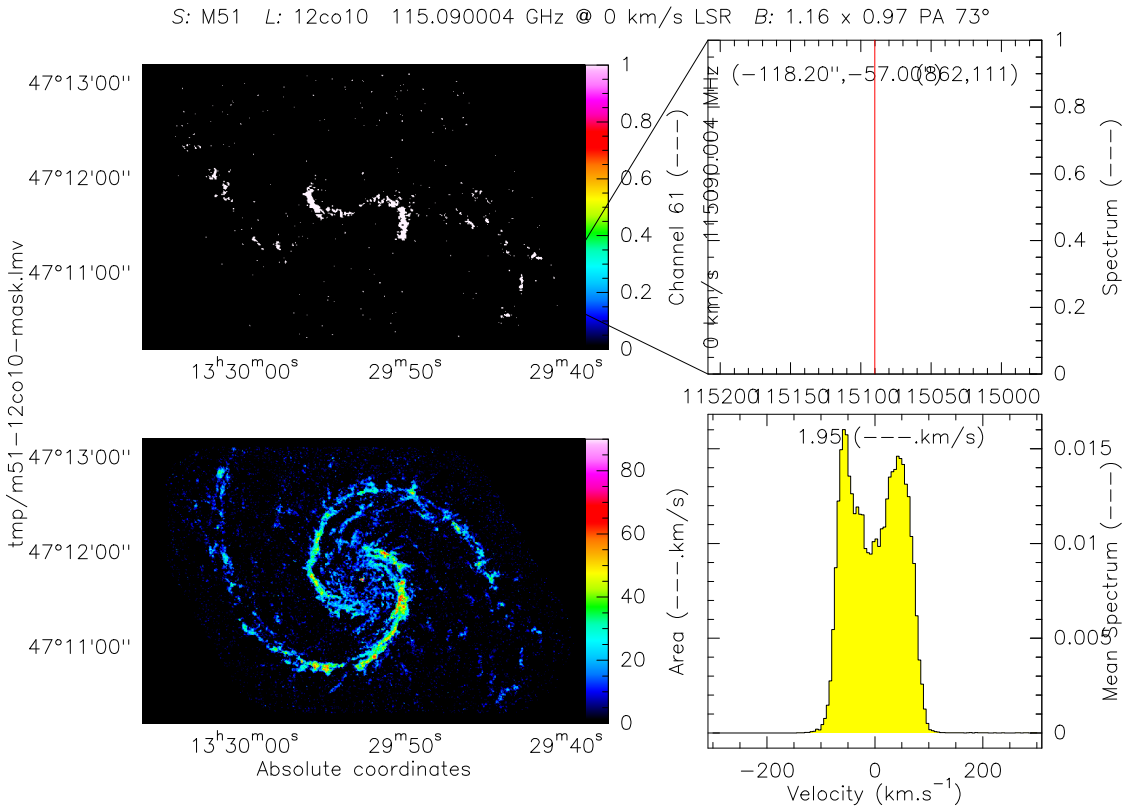
```
begin procedure moments-mask-define
  let name tmp/m51-12co10
  let type lmv
  ! Pass the SNR threshold used
  ! define the mask
  @ noise-mask-define 3
  ! Name set to tmp/m51-12co10-mask
  go view
end procedure moments-mask-define
!
@ noise-tools
@ moments-mask-define
```



Computing "moments": V. Finding noisy pixels

Procedure moments-mask-define in file pro/moments.greg

```
begin procedure moments-mask-define
  let name tmp/m51-12co10
  let type lmv
  ! Pass the SNR threshold used
  ! define the mask
  @ noise-mask-define 5
  ! Name set to tmp/m51-12co10-mask
  go view
end procedure moments-mask-define
!
@ noise-tools
@ moments-mask-define
```



Computing "moments": V. Finding noisy pixels

1. Checking the input parameters and loading the SNR cube

Procedure noise-snr in file tools/noise-tools.greg

```
if (pro%narg.ne.2) then
   message e noise-mask-define "Usage: @ noise-mask-define name value"
   return base
endif
let name "&1-snr"
@ cube-load snr
```

Computing "moments": V. Finding noisy pixels

2. Opening the output file, updating the header and the data, and updating the extrema

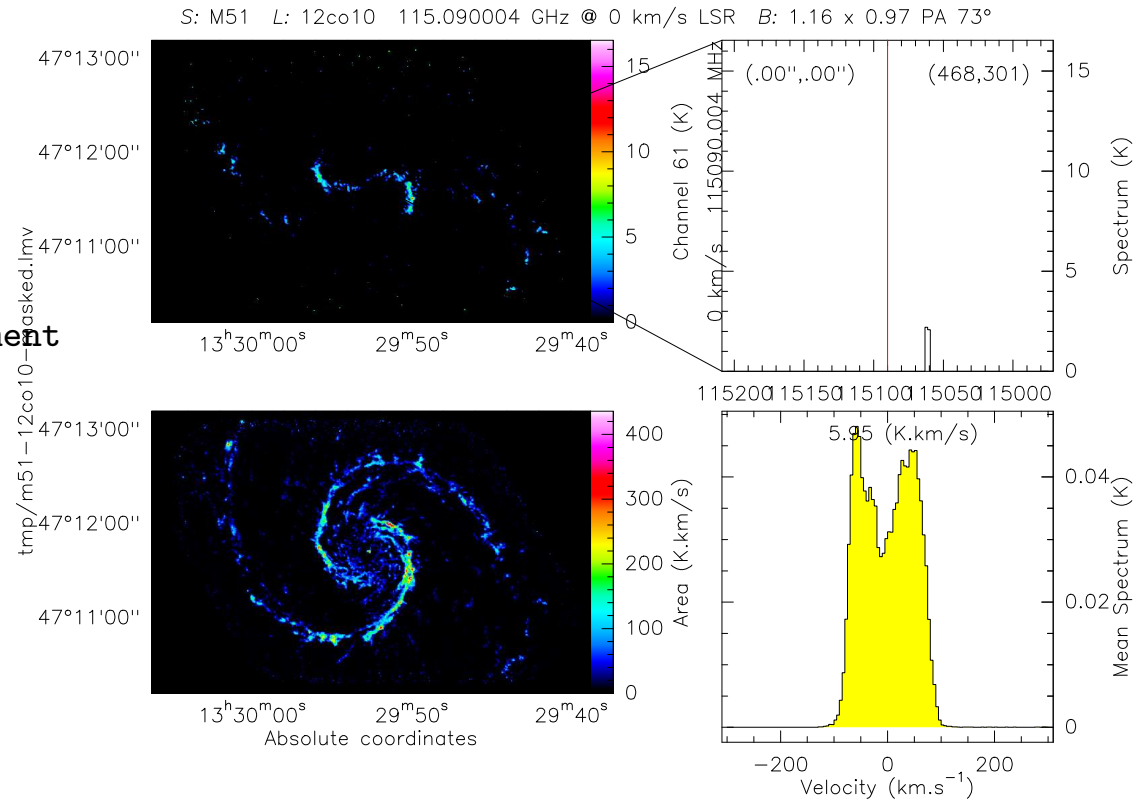
Procedure noise-snr in file tools/noise-tools.greg

```
define image mask "&1-mask."'type' real /like snr
let mask% snr%
! Mask will be 0 for SNR <= &2
! Mask will be 1 for SNR >  &2
let mask 0
let mask 1 /where snr.gt.&2
delete /var mask snr
let name "&1-mask"
message w noise-mask-define "Name set to "'name'
header 'name'"."'type' /extrema
```

Computing "moments": VI. Filtering noisy pixels

Procedure moments-mask-apply in file pro/moments.greg

```
begin procedure moments-mask-apply
  let name tmp/m51-12co10
  let type lmv
  ! Interpret the NAME variable and
  ! pass it as the procedure 1st argument
  @ noise-mask-apply 'name'
  ! Name set to tmp/m51-12co10-masked
  go view
end procedure moments-mask-apply
!
@ noise-tools
@ moments-mask-apply
```



Computing "moments": VI. Filtering noisy pixels

1. Checking the input parameters, and loading the signal and the mask cubes

Procedure noise-snr in file tools/noise-tools.greg

```
if (pro%narg.ne.1) then
   message e noise-mask-apply "Usage: @ noise-mask-apply name"
   return base
endif
let name "&1-mask"
@ cube-load mask
let name "&1"
@ cube-load sig
```

Computing "moments": VI. Filtering noisy pixels

2. Applying the mask to the signal cube, and updating the header, including the extrema

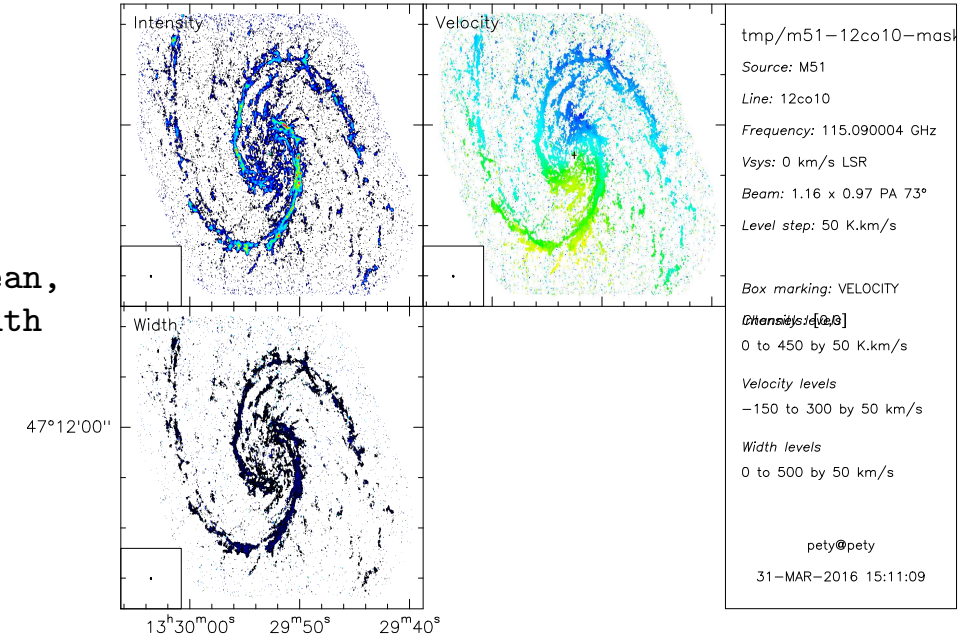
Procedure noise-snr in file tools/noise-tools.greg

```
define image masked "&1-masked."'type' real /like sig
let masked% sig%
let masked sig*mask
delete /var masked sig mask
let name "&1-masked"
message w noise-mask-apply "Name set to "'name'
header 'name'"."'type' /extrema
```

Computing "moments": VII. Actually computing the moments

Procedure moments-task in file pro/moments.greg

```
begin data gag_scratch:moments.init
  ! Moment.init
  TASK\FILE "Input file name" IN$ 'name'"."'type'
  TASK\FILE "Output files name (no extension)" OUT$ 'name'
  TASK\REAL "Velocity range" VELOCITY$[2] 'range[1]' 'range[2]'
  TASK\REAL "Detection threshold" THRESHOLD$ -1
  TASK\LOGICAL "Smooth before detection ?" SMOOTH$ no
  TASK\GO
end data gag_scratch:moments.init
!
begin procedure moments-task
  let name tmp/m51-12co10-masked
  let type lmv
  run moments gag_scratch:moments.init /nowin
  ! This task creates 3files: tmp/m51-12co10.mean,
  ! tmp/m51-12co10.velo, and tmp/m51-12co10.width
  ! Next command plot all of them together
  go velocity
end procedure moments-task
!
@ moments-task
```



Building the figure:
I. Loading an image from the disk to the RGDATA buffer

Procedure moments-plot-simple in file pro/moments.greg

```
begin procedure moments-plot-simple
  @ plot-simple-tools.greg
  !
  @ image-load noise
  Output: Filename tmp/m51-12co10-noi.lmv
  @ image-load peak
  Filename tmp/m51-12co10-peak.lmv
  @ image-load area
  Filename tmp/m51-12co10-area-31-90.lmv
  @ image-load mean
  Filename tmp/m51-12co10-masked.mean
  @ image-load velo
  Filename tmp/m51-12co10-masked.velo
  @ image-load width
  Filename tmp/m51-12co10-masked.width
end procedure moments-plot-simple
!
@ window_tools.greg
@ window_init
!
@ moments-plot-simple
```

Building the figure:
I. Loading an image from the disk to the RGDATA buffer

Procedure image-load in file tools/plot-simple-tools.greg

```
! Check input argument
if (pro%narg.ne.1) then
   message e image-load -
           "Usage: @ image-load nickname"
   return base
endif
```

```
! Translate nickname into the actual file name
define character nick*32 dir*128 postfix*128
let nick "&1"
let dir "tmp/"
let type "lmv"
let postfix ""
if (nick.eq."noise") then
   let postfix "-noi"
else if (nick.eq."peak") then
   let postfix "-peak"
else if (nick.eq."area") then
   let postfix "-area-31-90"
else if (nick.eq."mean") then
   let postfix "-masked"
   let type mean
else if (nick.eq."velo") then
   let postfix "-masked"
   let type velo
else if (nick.eq."width") then
   let postfix "-masked"
   let type width
else
   message e image-load "Unknown nickname: &1"
   return base
endif
let name 'dir'"m51-12co10"'postfix'
! User feedback
message r image-load "Filename "'name'"."'type'
```

Building the figure:
I. Loading an image from the disk to the RGDATA buffer

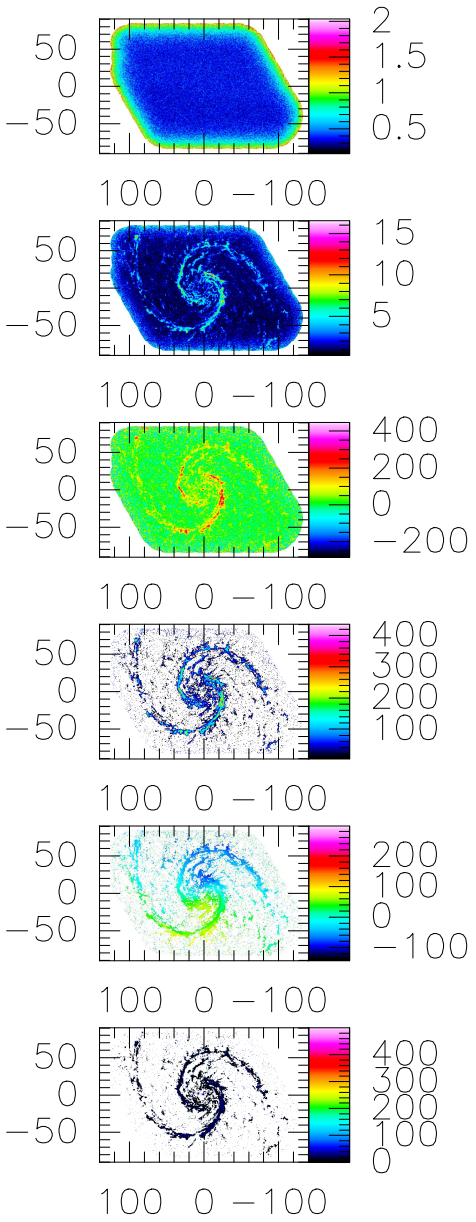
Procedure image-load in file tools/plot-simple-tools.greg

```
@ cube-load image ! Load file into the SIC IMAGE variable
rgdata image /var ! Load data into internal RG buffer
```

Building the figure: II. Controlling the page layout
1. Plotting 6 different images in one column

Procedure moments-plot-simple in file pro/moments.greg

```
begin procedure moments-plot-simple
  @ plot-simple-tools.greg
  !
  let inter 0.5 0.5
  @ image-layout 1 6
  @ image-plot-simple 1 noise
  @ image-plot-simple 2 peak
  @ image-plot-simple 3 area
  @ image-plot-simple 4 mean
  @ image-plot-simple 5 velo
  @ image-plot-simple 6 width
end procedure moments-plot-simple
!
@ window_tools.greg
@ window_init
!
@ moments-plot-simple
```

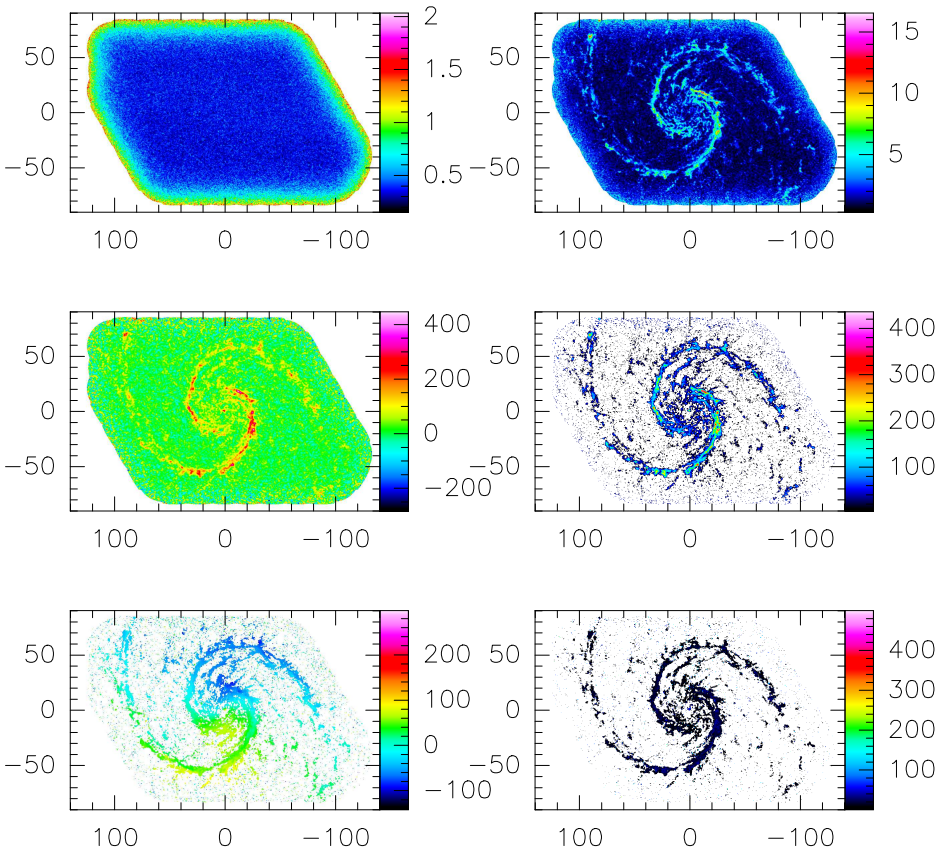


Building the figure: II. Controlling the page layout

2. Plotting 6 different images in two columns

Procedure moments-plot-simple in file pro/moments.greg

```
begin procedure moments-plot-simple
  @ plot-simple-tools.greg
  !
  let inter 0.5 0.5
  @ image-layout 2 3
  @ image-plot-simple 1 noise
  @ image-plot-simple 2 peak
  @ image-plot-simple 3 area
  @ image-plot-simple 4 mean
  @ image-plot-simple 5 velo
  @ image-plot-simple 6 width
end procedure moments-plot-simple
!
@ window_tools.greg
@ window_init
!
@ moments-plot-simple
```



Building the figure: II. Controlling the page layout

3. Defining the plot layout

Procedure image-layout in file tools/plot-simple-tools.greg

```
! Restart from a clean page
clear
! Check in the input parameters
if (pro%narg.ne.2) then
   message e image-load "Usage: @ image-layout nx ny"
   return base
endif
! Create global variables to transport information
! between procedures
if .not.exist(layout) then
   define structure layout /global
   define integer layout%nx layout%ny /global
endif
! Initialize the variables from the
let layout%nx &1
let layout%ny &2
! Here positivity of nx and ny could/should also be checked!
```

Building the figure: II. Controlling the page layout

4. Plotting a single image from a file

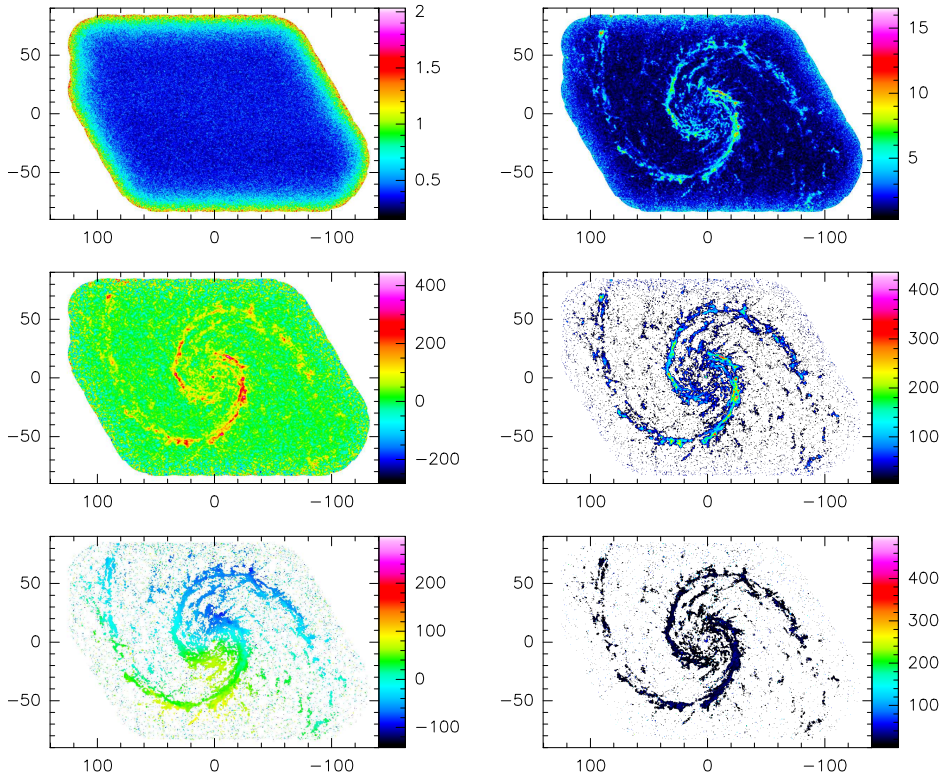
Procedure image-plot-simple in file tools/plot-simple-tools.greg

```
! Check in the input parameters
if (pro%narg.ne.2) then
   message e image-load "Usage: @ image-plot ith nickname"
   return base
endif
! Load image from file to RGDATA buffer (see previous slides)
@ image-load &2 ! Argument &2 is the image nickname
! Find image limits from RGDATA buffer
limits /rg
! Use possibility coded into window_tools.greg
let aspect_ratio yes ! Respect the aspect ratio defined in the ASPECT variable
let aspect abs((user_xmax-user_xmin)/(user_ymax-user_ymin))
! Define the panel position on page
@ window_xy 'layout%nx' 'layout%ny' &1
! Plot using a linear scale from the image minimum to the image maximum
plot /scaling lin 'image%min' 'image%max'
! Add the box labelled in arcsec (default is radian)
box /unit sec
! Add the color look-up table
wedge
```

Building the figure: III. Controlling the gaps between the images and the size of the characters

Procedure moments-plot-simple in file pro/moments.greg

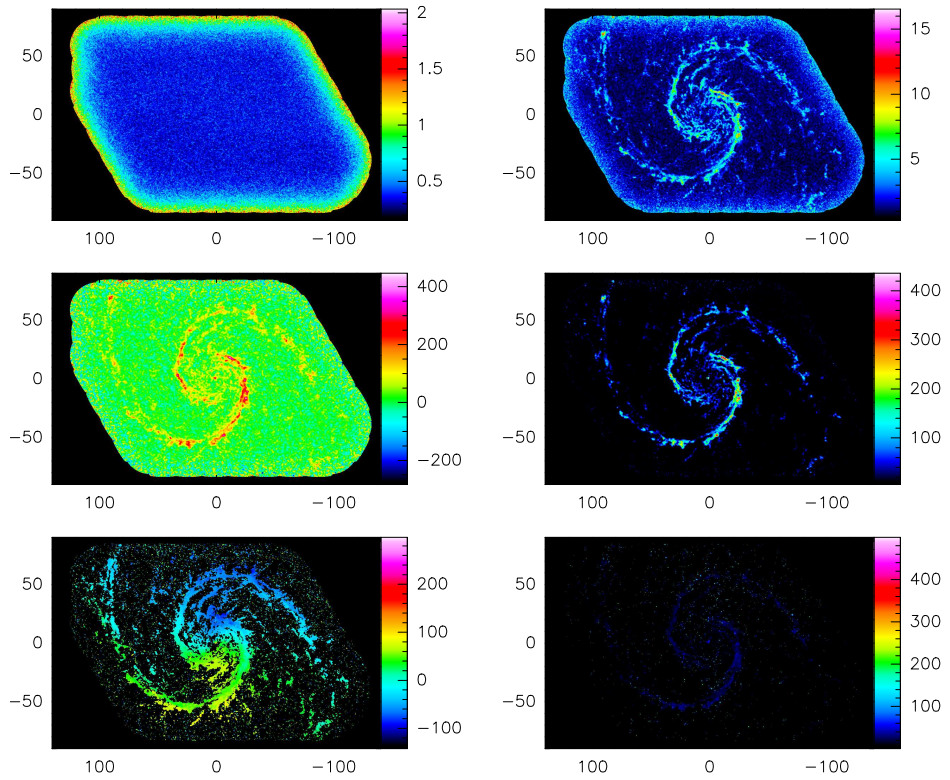
```
begin procedure moments-plot-simple
  @ plot-simple-tools.greg
  !
  set expand 0.7
  let inter 0.5 0.25
  @ image-layout 2 3
  @ image-plot-simple 1 noise
  @ image-plot-simple 2 peak
  @ image-plot-simple 3 area
  @ image-plot-simple 4 mean
  @ image-plot-simple 5 velo
  @ image-plot-simple 6 width
end procedure moments-plot-simple
!
@ window_tools.greg
@ window_init
!
@ moments-plot-simple
```



Building the figure:
IV. Changing the color associated to the blanking value

Procedure moments-plot-simple in file pro/moments.greg

```
begin procedure moments-plot-simple
  @ plot-simple-tools.greg
  !
  set expand 0.7
  let inter 0.5 0.25
  @ image-layout 2 3
  @ image-plot-simple 1 noise
  @ image-plot-simple 2 peak
  @ image-plot-simple 3 area
  @ image-plot-simple 4 mean
  @ image-plot-simple 5 velo
  @ image-plot-simple 6 width
  @ image-blank-black
end procedure moments-plot-simple
!
@ window_tools.greg
@ window_init
!
@ moments-plot-simple
```



Building the figure:
IV. Changing the color associated to the blanking value

Procedure image-blank-black in file tools/plot-simple-tools.greg

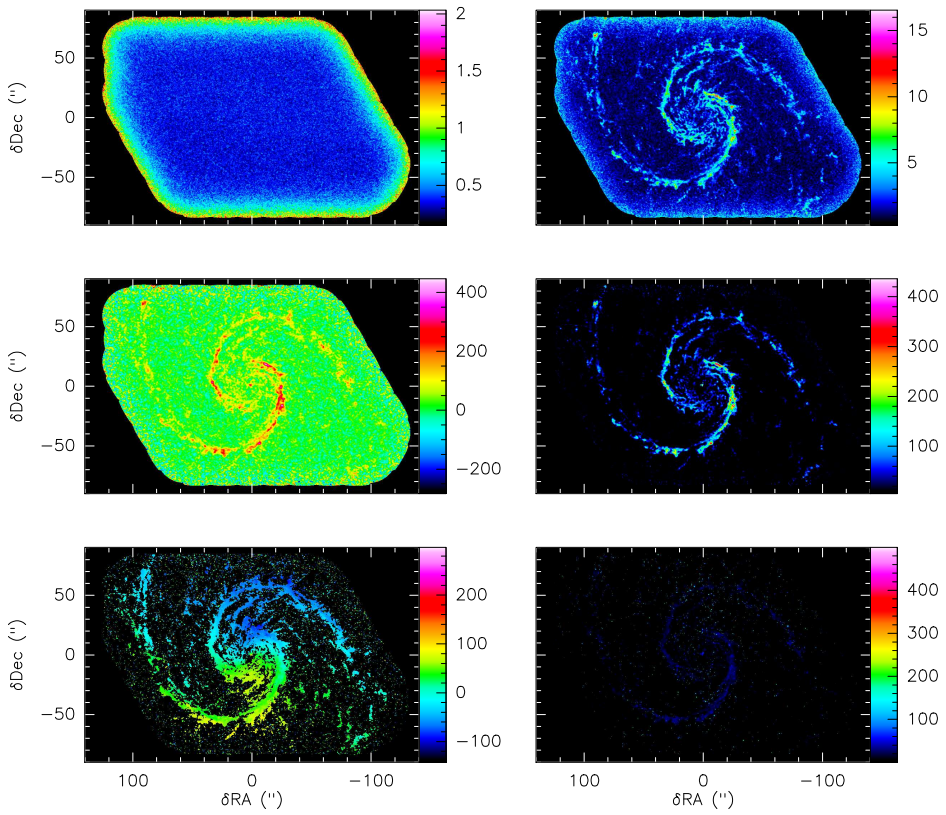
```
begin procedure image-blank-black
  ! Change lut mode from read-only to read-write
  let lut%mode 2
  ! Load black into blanking color buffer
  let lut%blank%red 0
  let lut%blank%blue 0
  let lut%blank%green 0
  ! Reload the lut to apply the user changes
  lut
end procedure image-blank-black
```

```
begin procedure image-blank-white
  ! Change lut mode from read-only to read-write
  let lut%mode 2
  ! Load white into blanking color buffer
  let lut%blank%red 256
  let lut%blank%blue 256
  let lut%blank%green 256
  ! Reload the lut to apply the user changes
  lut
end procedure image-blank-white
```

Building the figure: V. Cleverer boxes

Procedure moments-clever-box in file pro/moments.greg

```
begin procedure moments-plot-clever-box
  @ plot-clever-box-tools.greg
  !
  @ image-blank-black
  set expand 0.7
  let inter 0.35 0.25
  @ image-layout 2 3
  @ image-plot-clever-box 1 noise
  @ image-plot-clever-box 2 peak
  @ image-plot-clever-box 3 area
  @ image-plot-clever-box 4 mean
  @ image-plot-clever-box 5 velo
  @ image-plot-clever-box 6 width
  !
end procedure moments-plot-clever-box
!
@ window_tools.greg
@ window_init
!
@ moments-plot-clever-box
```



Building the figure: V. Cleverer boxes

Procedure image-plot-clever-box in file tools/plot-clever-box-tools.greg

Procedure image-box in file tools/plot-clever-box-tools.greg

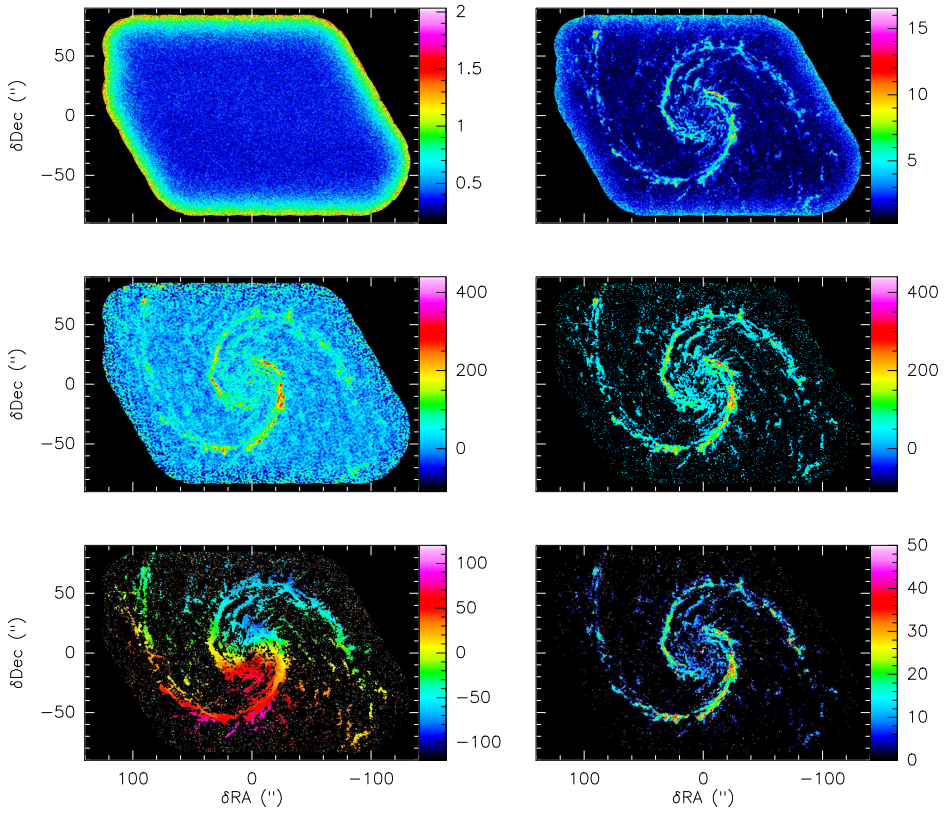
```
! Only a minor change in this procedure
begin procedure image-plot-clever-box
  if (pro%narg.ne.2) then
     message e image-load -
             "Usage: @ image-plot ith nickname"
     return base
  endif
  @ image-load &2
  limits /rg
  let aspect_ratio yes
  let aspect -
      abs((user_xmax-user_xmin)/(user_ymax-user_ymin))
  @ window_xy 'layout%nx' 'layout%ny' &1
  plot /scaling lin 'image%min' 'image%max'
  @ image-box &1
  wedge
end procedure image-plot-clever-box
```

```
begin procedure image-box
  ! Check input parameters
  if (pro%narg.ne.1) then
     message e image-load "Usage: @ image-box ith"
     return base
  endif
  ! Put y axis labels only for right panels
  if (('mod(&1,layout%nx)'.eq.1).or. -
      (layout%nx.eq.1)) then
     axis yl /label o /unit sec
     label "\\Gd\\RDec (')" /y 4.5
  endif
  ! Put x axis labels only for bottom panels
  if (&1.gt.'layout%nx*(layout%ny-1)') then
     axis xl /label p /unit sec
     label "\\Gd\\RRA (')" /x
  endif
  ! Overlay black box with white ticks
  ! and no labels
  pen 7
  box n n /unit sec
  pen 0
  box p o n /unit sec
end procedure image-box
```

Building the figure: VI. Adapting the color scale

Procedure moments-custom-scale in file pro/moments.greg

```
begin procedure moments-plot-custom-scale
  @ plot-custom-scale-tools.greg
  !
  @ image-blank-black
  set expand 0.7
  let inter 0.35 0.25
  @ image-layout 2 3
  let scale 0 0 ! Default linear scale
  @ image-plot-custom-scale 1 noise
  @ image-plot-custom-scale 2 peak
  let scale -110 440
  @ image-plot-custom-scale 3 area
  @ image-plot-custom-scale 4 mean
  let scale -120 +120
  @ image-plot-custom-scale 5 velo
  let scale 0 50
  @ image-plot-custom-scale 6 width
  !
end procedure moments-plot-custom-scale
!
@ window_tools.greg
@ window_init
!
@ moments-plot-custom-scale
```



Building the figure: VI. Adapting the color scale

Procedure image-plot-custom-scale in file tools/plot-custom-scale-tools.greg

```
begin procedure image-plot-custom-scale
  if (pro%narg.ne.2) then
     message e image-load "Usage: @ image-plot ith nickname"
     return base
  endif
  @ image-load &2
  limits /rg
  let aspect_ratio yes
  let aspect abs((user_xmax-user_xmin)/(user_ymax-user_ymin))
  @ window_xy 'layout%nx' 'layout%ny' &1
  @ image-scale
  plot /scaling lin 'ima%scale[1]' 'ima%scale[2]'
  @ image-box &1
  wedge
end procedure image-plot-custom-scale
```

Building the figure: VI. Adapting the color scale

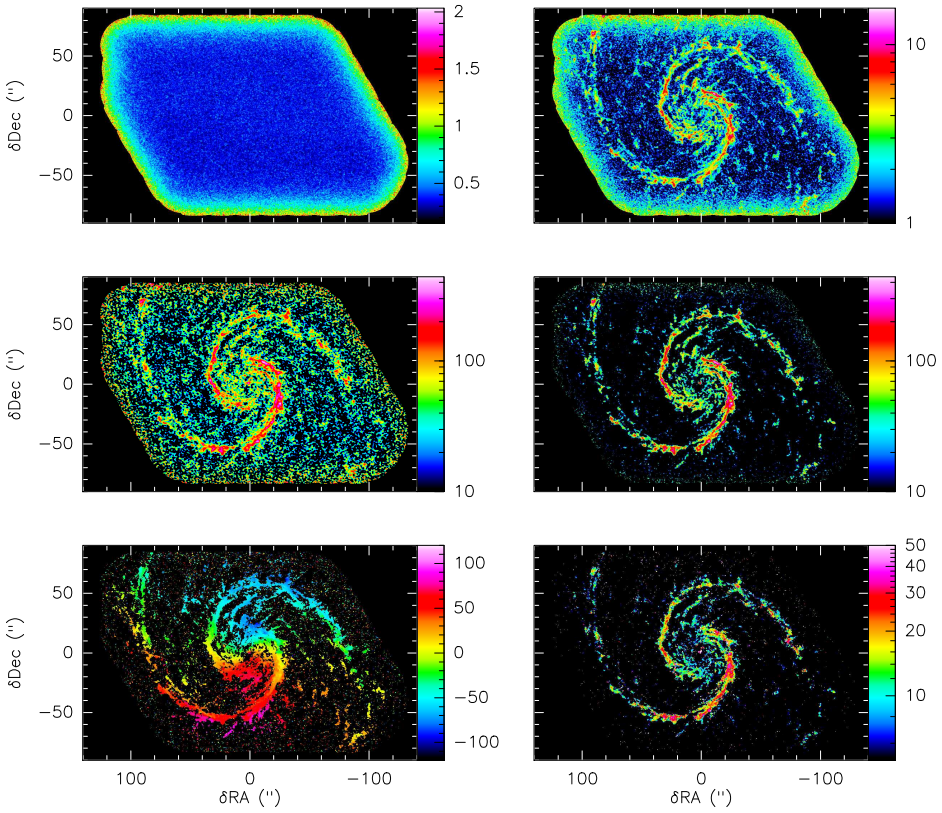
Procedure image-scale in file tools/plot-custom-scale-tools.greg

```
begin procedure image-scale
  ! Create global variables for inter-procedure communication
  if .not.exist(ima) then
     define structure ima /global
  endif
  if .not.exist(ima%scale) define double ima%scale[2] /global
  !
  if ((scale[1].eq.0).and.(scale[2].eq.0)) then
    ! Default => Image minimum and maximum
     let ima%scale 'image%min' 'image%max'
  else
    ! User defined
     let ima%scale scale
  endif
  message r image-scale "Scale set to ["'ima%scale[1]'"-"'ima%scale[2]'"]"
end procedure image-scale
```

Building the figure: VII. Linear or logarithmic color scale

Procedure moments-linlog-scale in file pro/moments.greg

```
begin procedure moments-plot-linlog-scale
  @ plot-linlog-scale-tools.greg
  !
  @ image-blank-black
  set expand 0.7
  let inter 0.35 0.25
  @ image-layout 2 3
  @ image-lin  0 0   ! Default linear scale
  @ image-plot-linlog-scale 1 noise
  @ image-log  1 16  ! Logarithmic scale
  @ image-plot-linlog-scale 2 peak
  @ image-log 10 440 ! Logarithmic scale
  @ image-plot-linlog-scale 3 area
  @ image-plot-linlog-scale 4 mean
  @ image-line -120 +120 ! Default linear scale
  @ image-plot-linlog-scale 5 velo
  @ image-log 5 50   ! Logarithmic scale
  @ image-plot-linlog-scale 6 width
  !
end procedure moments-plot-linlog-scale
!
@ window_tools.greg
@ window_init
!
@ moments-plot-linlog-scale
```



Building the figure: VII. Linear or logarithmic color scale

Procedure image-scaling in file tools/plot-linlog-scale-tools.greg

Procedure image-lin in file tools/plot-linlog-scale-tools.greg

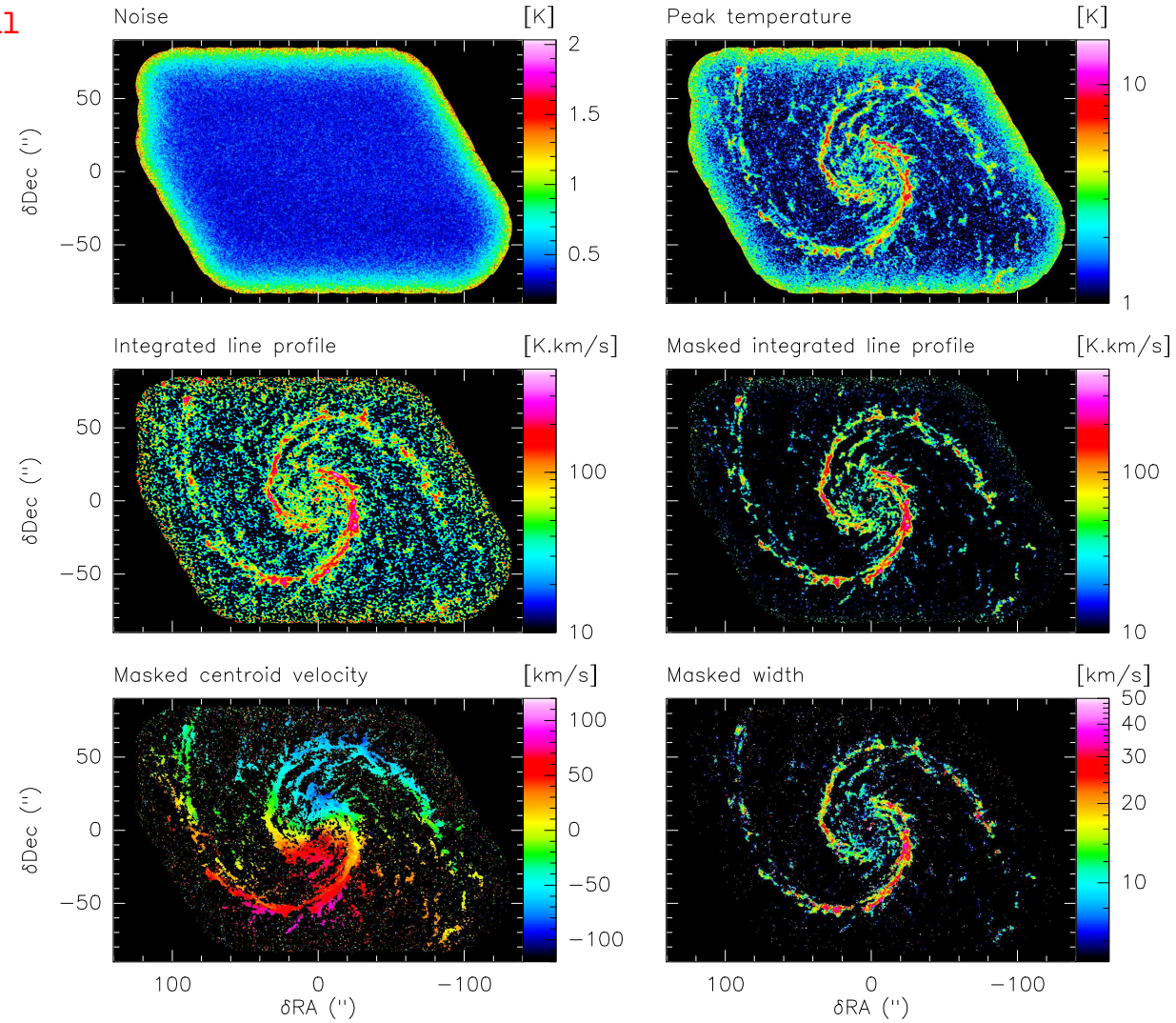
Procedure image-log in file tools/plot-linlog-scale-tools.greg

```
begin procedure image-scaling
  ! Create global variables for inter-procedure communication
  if .not.exist(ima) then
     define structure ima /global
  endif
  if .not.exist(ima%scaling) define character ima%scaling*3 /global
  ! Set the created variable with first procedure argument
  let ima%scaling "&1"
end procedure image-scaling
!
begin procedure image-lin
  ! Create and set the IMA%SCALING global variable
  @ image-scaling lin
  ! Set the existing global variable SCALE with procedure arguments
  let scale &1 &2
end procedure image-lin
!
begin procedure image-log
  @ image-scaling log
  let scale &1 &2
end procedure image-log
```

Building the figure: VIII. Title and unit

Procedure moments-plot-final in file pro/moments.greg

```
begin procedure moments-plot-final
  @ plot-tools.greg
  !
  @ image-blank-black
  set expand 0.7
  let inter 0.35 0.25
  @ image-layout 2 3
  @ image-lin 0 0
  @ image-plot 1 noise
  @ image-log 1 16
  @ image-plot 2 peak
  @ image-log 10 440
  @ image-plot 3 area
  @ image-plot 4 mean
  @ image-lin -120 +120
  @ image-plot 5 velo
  @ image-log 5 50
  @ image-plot 6 width
  !
end procedure moments-plot-final
!
@ window_tools.greg
@ window_init
!
@ moments-plot-plot-final
```



Building the figure: VIII. Title and unit
1. Consolidating the use of global variables

Procedure image-init in file tools/plot-tools.greg

```
begin procedure image-init
  ! Definition of global variables is factorized here
  if .not.exist(ima) then
     define structure ima /global
     define integer ima%nx ima%ny /global
     define character ima%scaling*3 ima%title*32 ima%unit*32 /global
  endif
    ! Default to automatic linear scaling
  @ image-lin 0
end procedure image-init
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
@ cube-tools.greg
@ window_tools.greg
! Initialize the image-tools
@ image-init
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Building the figure: VIII. Title and unit

2. Drawing the title and the unit

Procedure image-plot in file tools/plot-tools.greg
Procedure image-unit in file tools/plot-tools.greg
Procedure image-title in file tools/plot-tools.greg

```
begin procedure image-plot
  ! Only minor changes in this procedure
  if (pro%narg.ne.2) then
     message e image-plot -
             "Usage: @ image-plot ith nickname"
     return base
  endif
  @ image-load &2
  limits /rg
  let aspect_ratio yes
  let aspect -
      abs((user_xmax-user_xmin)/(user_ymax-user_ymin))
  @ window_xy 'layout%nx' 'layout%ny' &1
  plot /scaling lin 'image%min' 'image%max'
  @ image-box &1
  @ image-title
  @ image-unit
  wedge
end procedure image-plot
!
begin procedure image-title
  draw text 0 1 'ima%title' 6 0 /char 7
end procedure image-title
!
begin procedure image-unit
  draw text 0 1 "['ima%unit'"]" 6 0 /char 9
end procedure image-unit
```