12m Spectral Line Unipops Examplebook

Version 1.0

Contents

contents 5/3/95

rms, peak, crosshair, crossflg, bmoment, emoment, moment, size, tcur, vcur, ccur, gset, gauss, gparts, gdisplay, residual, rline, reshow, batch, tilesaves, nbox, hanning, boxcar, chngres, newres

rampup, scandist, scanrate, #rows, spacing

tell, nsave, save, copy, recall, sprotect, keep, kprotect, kget, check

uni2fits, eqtogal, galtoeq, system, page, show, tcopy, gcopy, get, header, default, read, tell, setenv DISPLAY hostname:0, xhost, stty erase ^H, tell disk, dscans, kscans, sscans ondata, offdata, gscans, chngver, chngonnline, summary, cget, select, spike, gget, cget

Unipops Examplebook for Spectral Line Users of the 12 m Telescope Version 1.0

by Kathryn N. Mead

Introduction

Who Is This Book For?

This book is written for unipops users who are observing and thus don't have the time or patience to study the *Unipops Cookbook* (UC). For the rookie, it is meant to provide the basic commands needed for on-line data reduction. For the veteran, it is meant to remind you of familiar commands, or help you find the new words for old favorites. (Sorry, no old-to-new dictionary.) Everything else you figure out yourself. Smart, computer literate astronomers don't need some dumb manual to tell you stuff that you have already figured out.

How To Use This Book

Unipops is a spectra based data reduction program and therefore this document is a collection of figures. The spectra in this manual were produced in unipops (line) by the commands shown to the right of each spectrum. These are the commands to type at the LineF> prompt. In the first few pages, the prompt and all of the commands (starting with `f` or `s`) needed to produce the spectrum are shown. Later, only the commands to get that particular illustrated effect are shown.

`User typed commands` are printed in **`courier bold`** font. Annotations to the commands are printed in Times Roman font (this font.) **Computer responses** to user commands are printed in **helvitica** because that's the font that is currently used by Line. Users of the 12m should be able to easily locate commands either by thumbing through the pictures, scanning the table of contents, or by using the index. The commands used in each chapter are summarized at the end of the chapter listing on the contents page.

On Line Help

`whatis` *`commandname`* tells you what part of speech a command is, verb, adverb, psuedo-verb, or procedure. You don't need to know this to use the command, so whatis probably won't be too useful to you. If you are going to do something sophisticated with unipops, like program in it, then you will start to be concerned with these parts of speech. In that case, you are more advanced than this manual.

`help` *`command-name`* gives a one line description of all commands except procedures. Ok, so you need to know what a procedure is... For now, you only need to know that it's a command for which there is no help. If you don't get satisfaction from help, try explain.

`explain` *`commandname`* spawns a popup window wit h a VMS like help facility which contains perhaps useful, perhaps too sophisticated information. Explain tells you something about how a command works.

`identify` *`keyword search-level`* prints, in the text window, all commands related to keyword. Search level is 0 for "fast search" or 1 for "complete" search. This is useful when you know what you want to do but can't remember the right word. However, because the commands are printed in alphabetical order, there is no way to quickly access the more rudimentary commands related to the topic/keyword that you are interested in. Syntax: you can type, for example, `identify baseline 0`, or just `baseline` and it will query you for the other inputs.

I have found `help` and `explain` to be useful when I have the time and inclination to pay attention. (i.e. don't invoke explain while you are reducingt 1 minute scans in real time!) `identify` returns an enormous number of commands in some cases. So many that this command may be of limited use for many users.

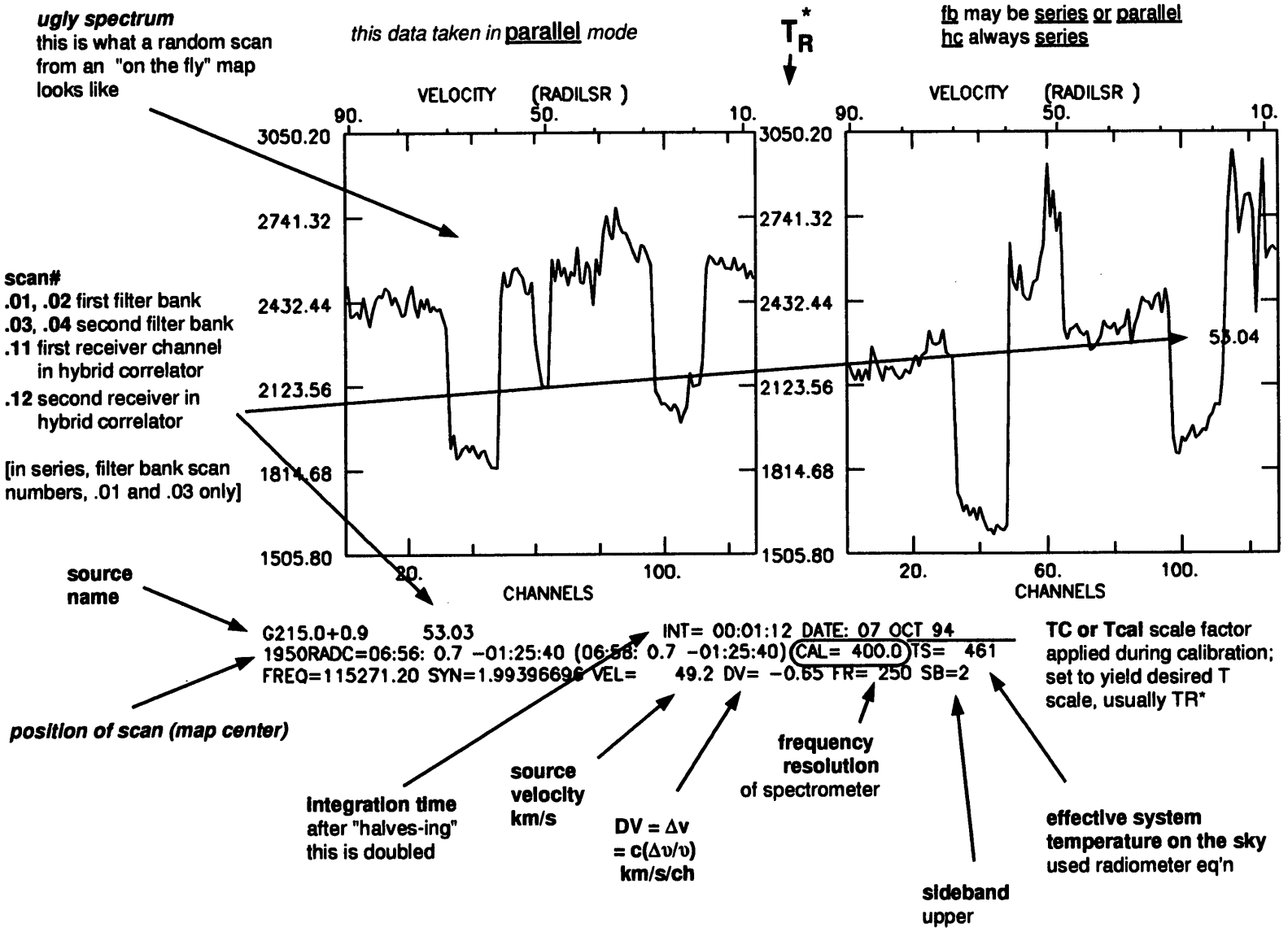
Unipops Cookbook

In working on this picturebook, I have found that there is a wealth of information in the *Unipops Cookbook*. However, because of the way it is arranged (from a programmers perspective) and presented (miniscule type and no figures), it takes some patience and concentration to assimilate the information. While I'm observing, my attention span is too short to really make the best use of the UC. However, at some point, you will find that this picturebook is entirely too elementary. When this happens, turn to the UC.

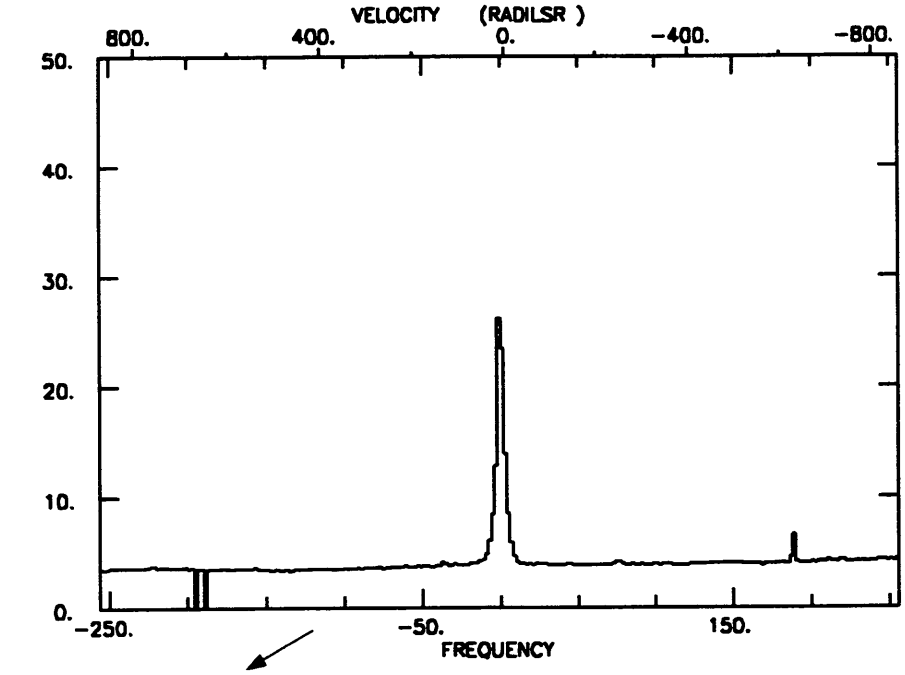
How To Make This Book Better

If something about this manual makes you want to throw it across the room, please tell me what it is; I will improve it and save the next observer from the same urge. If there's a feature that you like, tell me that too so I can be sure to keep it. I also collect mistake reports. I have done my own share of griping about documentation (I think that's how I got this job :-)), so don't be shy! reach me at kmead@nrao.edu

The *report facility* is also useful for reaching the programmer and other interested people. Simply type `report,` and a window with your favorite editor will be spawned (after a short delay). Just type your message and exit in the usual way and an e-mail will be sent to the distribution list. This is the best way to communicate with the programmer. He really reads messages produced in this way; they do not go into a black hole!


ugly spectrum
this is what a random scan from an "on the fly" map looks like
this data taken in parallel mode
T_R^*
fb may be series or parallel
hc always series
VELOCITY (RADILSR)
90.
50.
10.
3050.20
2741.32
2432.44
2123.56
1814.68
1505.80
20.
60.
100.
CHANNELS
53.04
scan#
.01, .02 first filter bank
.03, .04 second filter bank
.11 first receiver channel in hybrid correlator
.12 second receiver in hybrid correlator
[in series, filter bank scan numbers, .01 and .03 only]
source name
G215.0+0.9 53.03
1950RADC=06:56: 0.7 -01:25:40 (06:56: 0.7 -01:25:40)
INT= 00:01:12 DATE: 07 OCT 94
CAL= 400.0 TS= 461
FREQ=115271.20 SYN=1.99396696 VEL= 49.2 DV= -0.65 FR= 250 SB=2
position of scan (map center)
integration time after "halves-ing" this is doubled
source velocity km/s
DV = Δv = c(Δυ/υ) km/s/ch
frequency resolution of spectrometer
sideband upper
TC or Tcal scale factor applied during calibration; set to yield desired T scale, usually TR*
effective system temperature on the sky used radiometer eq'n

Calling up Series Data

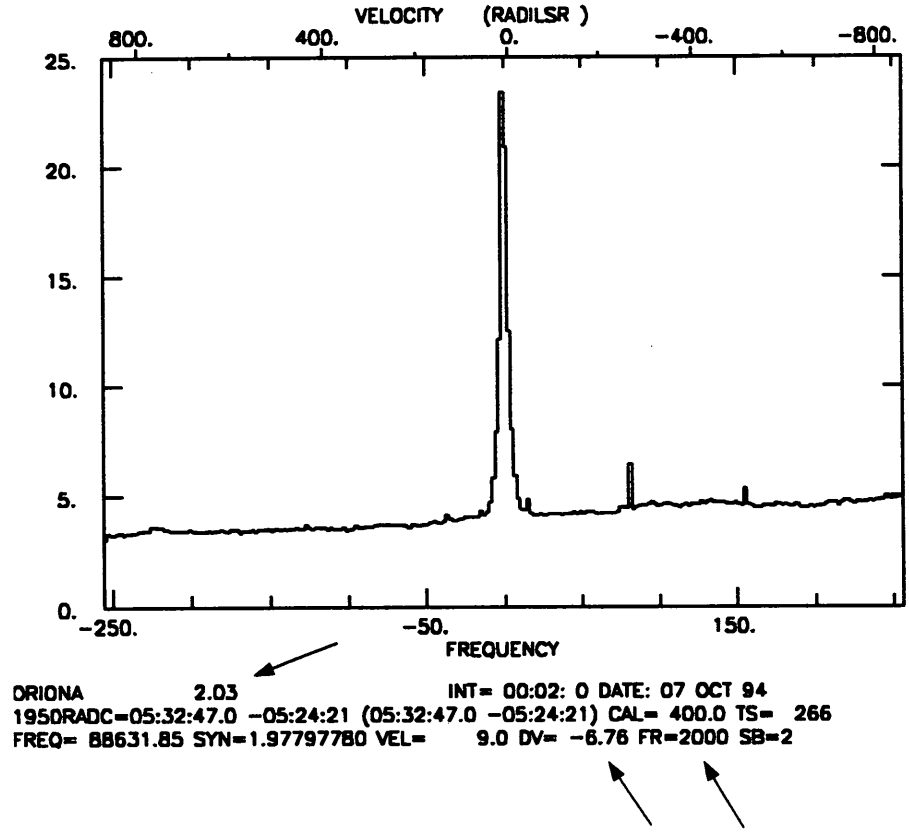


LineF>**2f**

Arrows point to scan #, frequency resolution and velocity resolution. Notice that in series, the subscan numbers are .01 and .03.

f. s
make
hardcopy

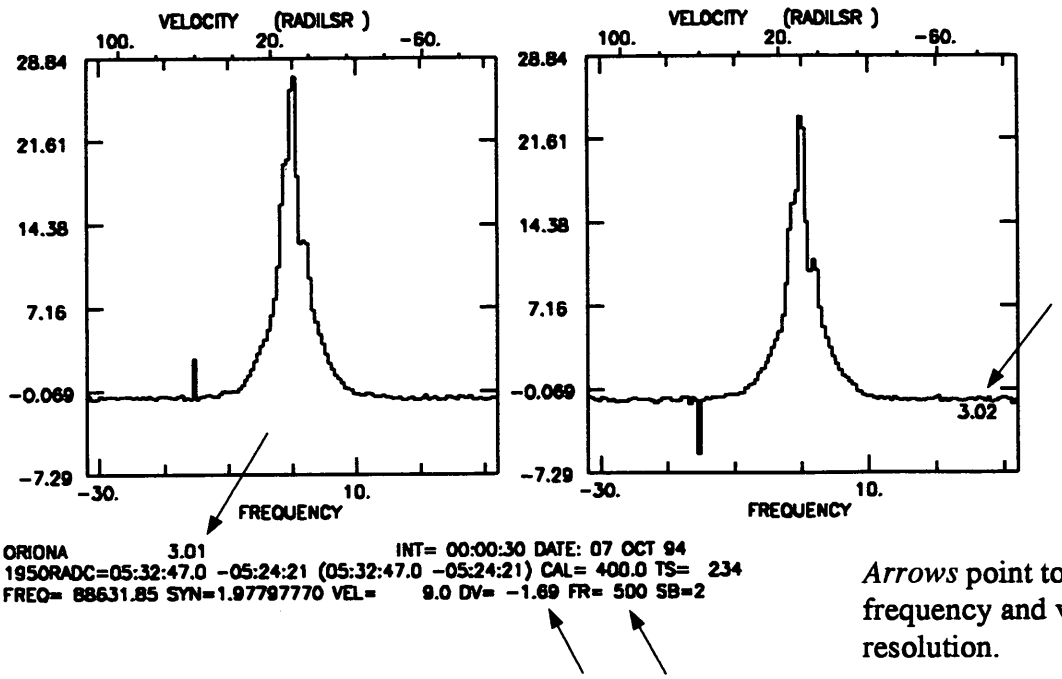
Usually, series mode is used instead of parallel mode due to bandwidth requirements. (This is not a good example of use of series mode!) So, often, the two 1MHz or 2MHz filter banks are used in series. That way, the data from the two receivers can be averaged.



LineF>**2s**

LineF>**make**
makes hardcopy

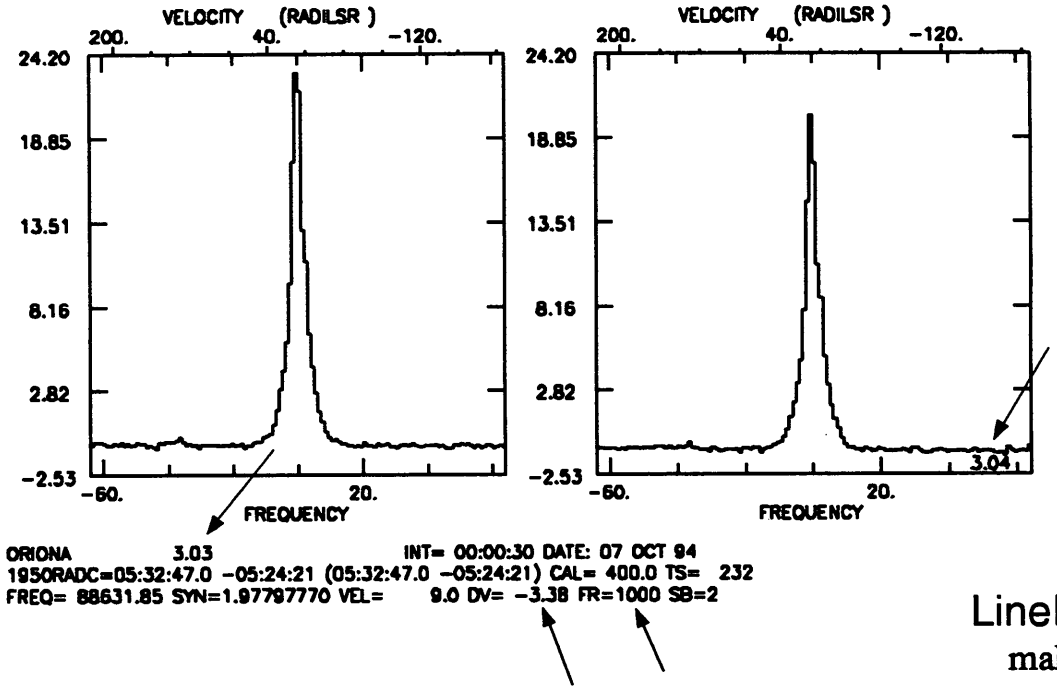
Calling up Parallel Data



LineF>**3f**

Arrows point to scan #'s and frequency and velocity resolution.

In parallel mode, frist filter bank has scan numbers, .01 and .02 (first and secnd receiver, respectively.) Second filter bank has subscan numbers .03 and .04.



LineF>**3s**

LineF>**make**
makes hardcopy

```
LineF>get 36 header
 ---------------------------------------------------------------------
 SCAN  36.01   NRAO 12M     G93.2+1.7     mlk
 ---------------------------------------------------------------------
 MM/DD/YY          LST          UTC          MODE   OPR
 01/11/95        18:48: 1.0   18:51:13.7      PS     HDS
          RA       DEC                    OFFSETS
                                        POINTING  +BEAM    REF
 1950         21:16:33.3  51:39:11    AZ:   -0:05   02:00    30:00
 1950 REF     21:16:39.8  51:38:11    EL:   00:14   00:00    00:00
 GALACTIC      93.2351     1.7117     RA:   -1:00             00:00
 AZ/EL         43.7568    56.3274    DEC:   01:00             00:00
  TSYS   TC   RCVRS  BW/CHAN  REST FREQ   DOP FREQ   IFS   SB
   424. 400.0   1      100.0     146969.048  147005.029  100.  2.
                                 1.96358148              1500.
  TIME  NS   SEC     VEL     VLSR/VR  VEL/CHAN  TOL    F0
   6.0  12    30.0   -71.00  -2.395    -0.204     5.   48.0
LineF>tcopy
```

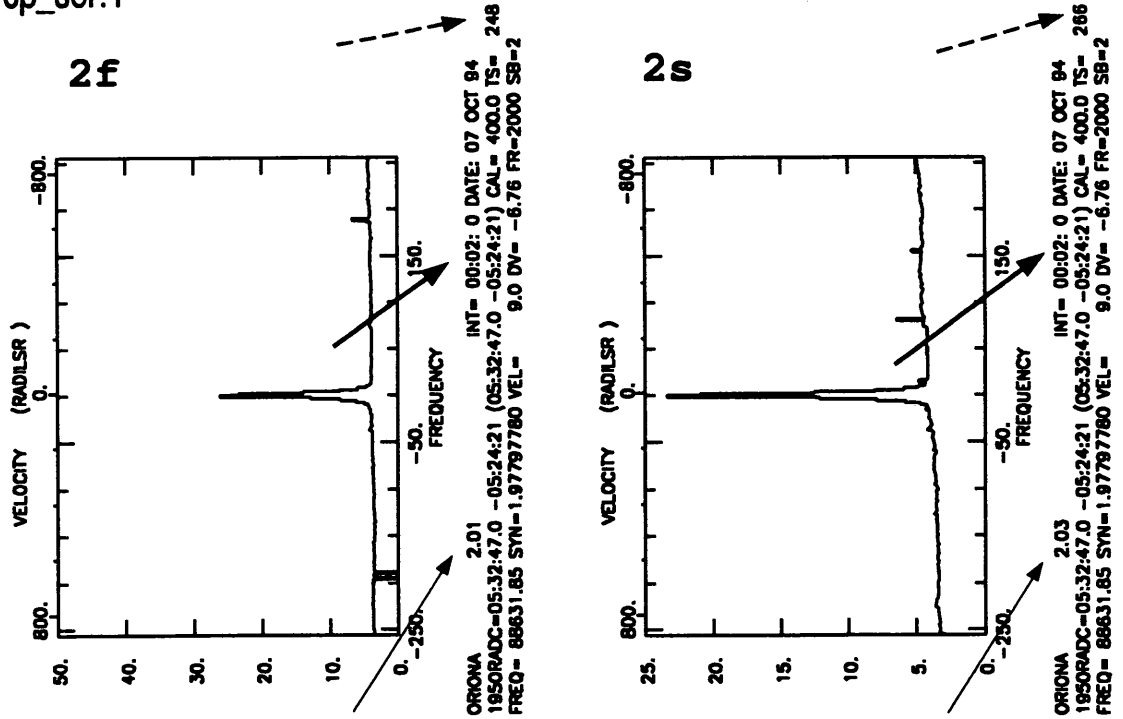
get calls up the scan but does not plot it

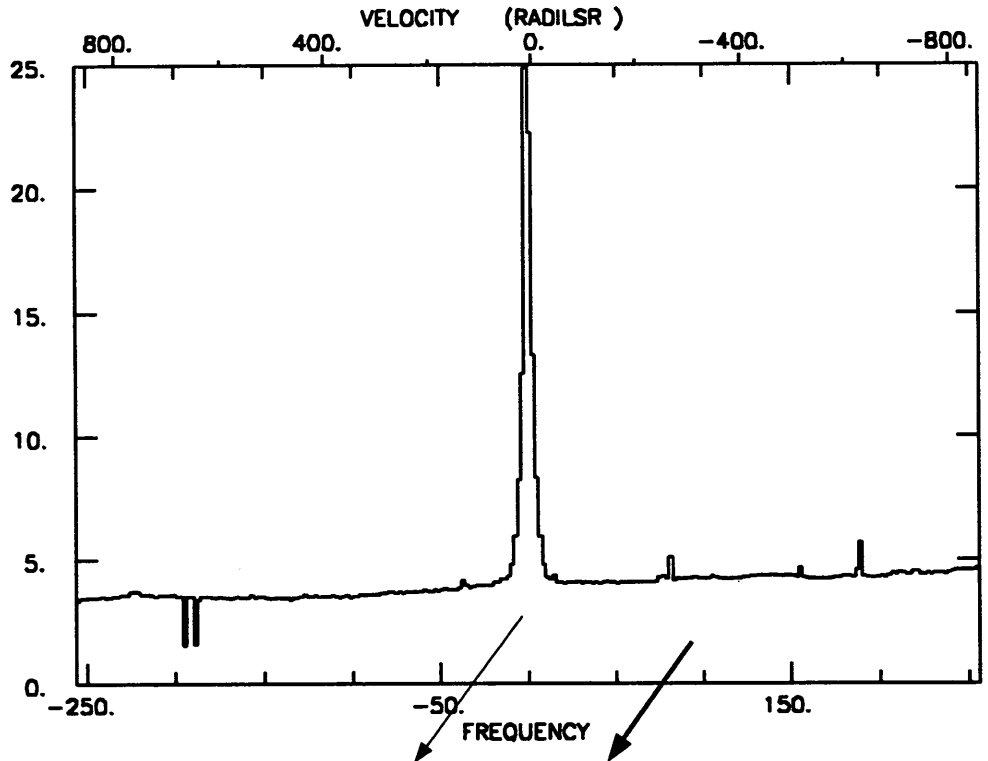
header displays, in the text window, the information shown above

tcopy prints the last 20 lines of text

Source offsets are shown in the "offsets", "pointing" column, in the RA, DEC row, so this scan was done at the (1'W,1'N) position. In this example, the "1950 REF" coordinates are the coordinates in the catalog, and the "1950" coordinates are the sum of the "RA DEC Pointing Offsets" + the "1950 REF " coordinates.

cp_ser.1





empty

empty the stack

2 a

put scan 2 in the stack

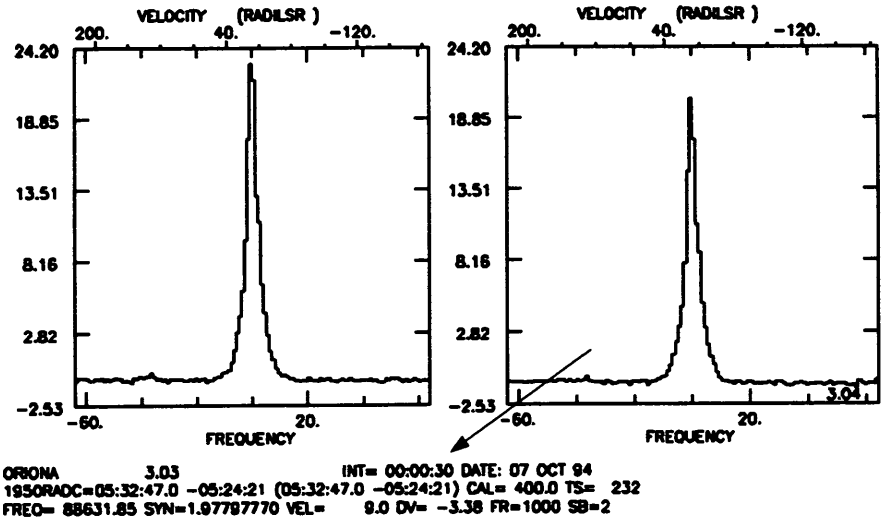
cb

"combine both" (.01&.03 scans)

```
ORIONA          2 SCANS:    2.01-     2.03 INT= 00:04: 0 DATE: 07 OCT 94
1950RADC=05:32:47.0 -05:24:21 (05:32:47.0 -05:24:21) CAL= 400.0 TS=  257
FREQ= 88631.85 SYN=1.97797780 VEL=      9.0 DV= -6.76 FR=2000 SB=2
```

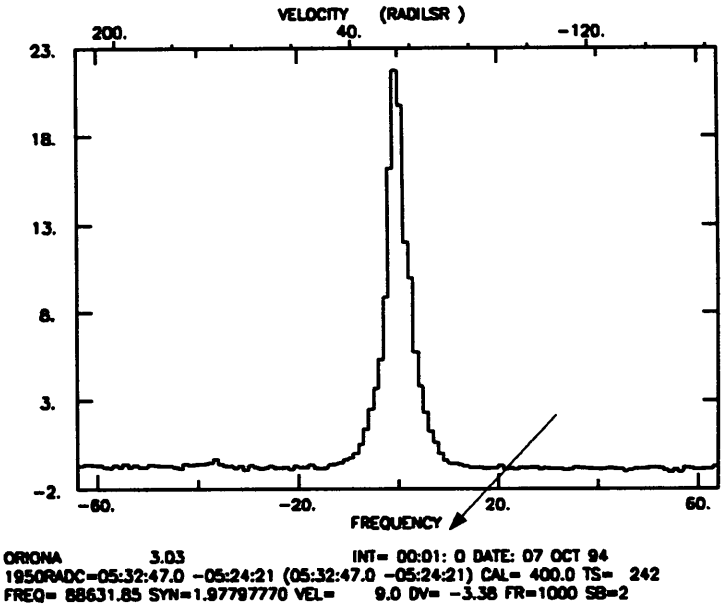
notice how scan#, integration time, and system temperature change appropriately

Combining Polarizaions in Parallel Data



halves

3s

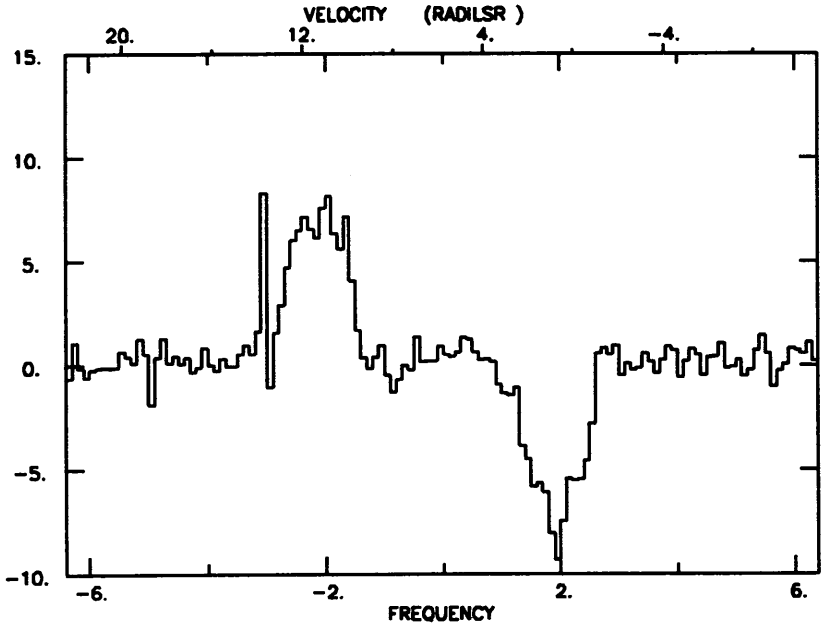


halves
averages the above two scans together

arrows point out how integration time changes appropriately

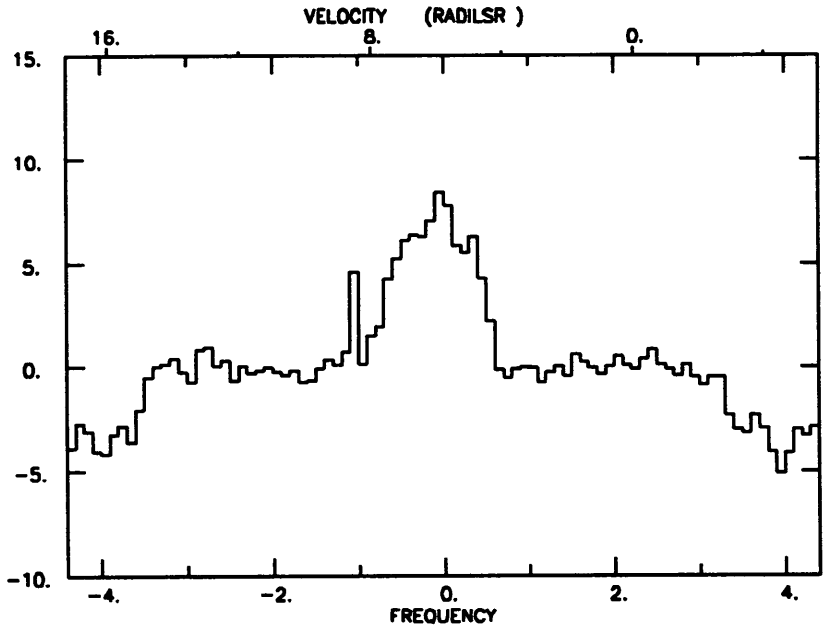
cp_par.1

Folding Frequency Switched Data



TMC1 23.01 INT= 00:00:20 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 473
FREQ=115271.20 SYN=1.99429034 VEL= 5.8 DV= -0.26 FR= 100 SB=2

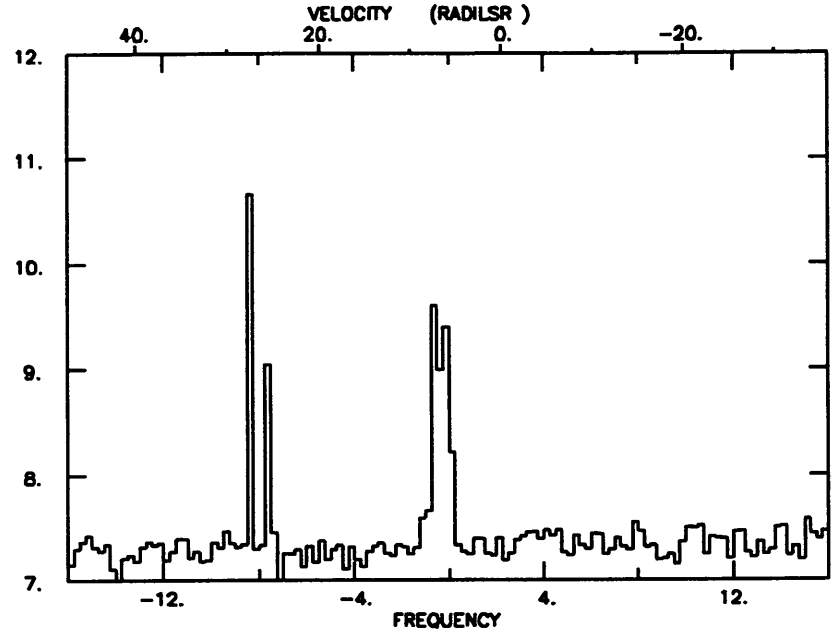
23f halves



TMC1 23.01 INT= 00:00:20 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 473
FREQ=115271.20 SYN=1.99429034 VEL= 5.8 DV= -0.26 FR= 100 SB=2

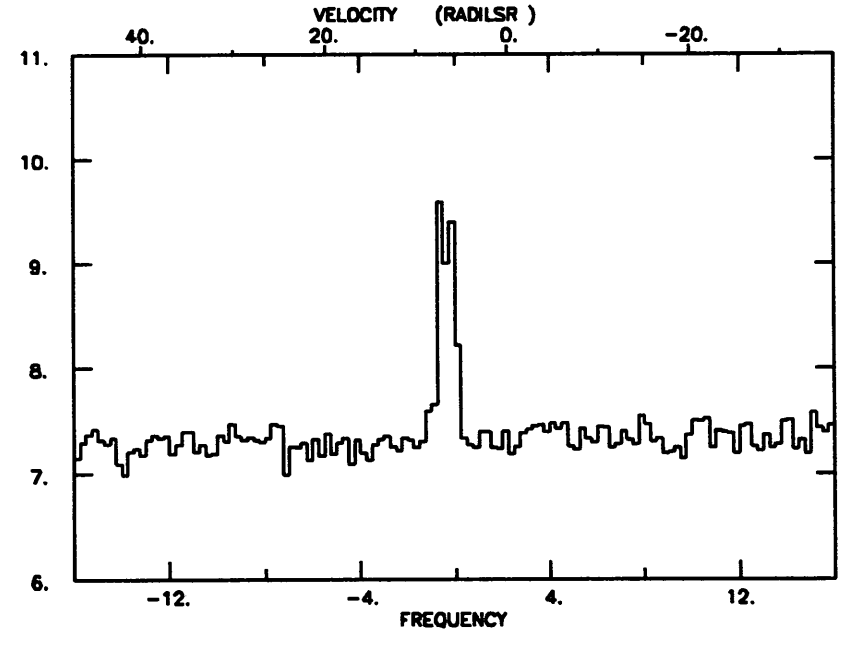
fold

Removing Bad Channels 1



TMC1 19.01 INT= 00:04: 0 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 448
FREQ=115271.20 SYN=1.95990626 VEL= 5.8 DV= -0.65 FR= 250 SB=2

19f halves

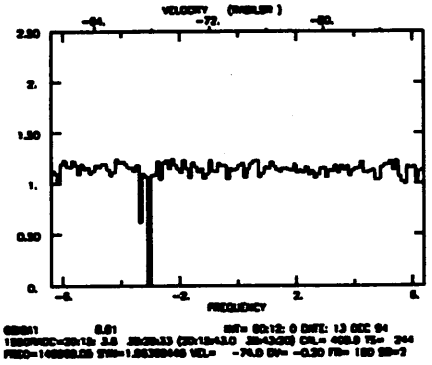


TMC1 19.01 INT= 00:04: 0 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 448
FREQ=115271.20 SYN=1.95990626 VEL= 5.8 DV= -0.65 FR= 250 SB=2

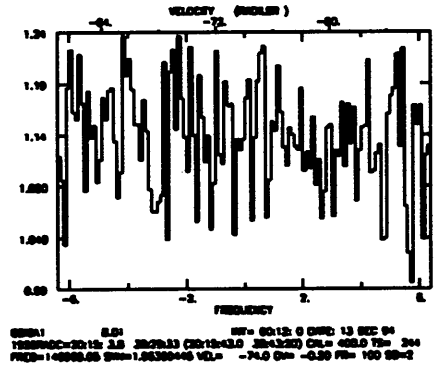
badch

answer question
and
follow mousing
instructions

Removing Bad Channels 2

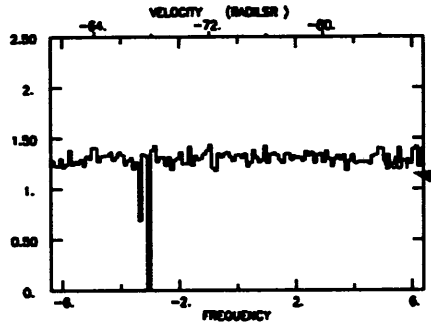


8f halves



badch

this command places the identified channels in array badpt

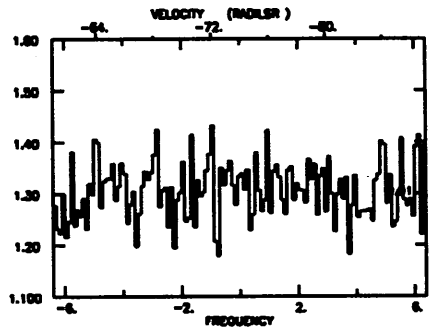


9f

slabel=2

you can't see it, but the scan number, 9.01, is there in the noise

halves

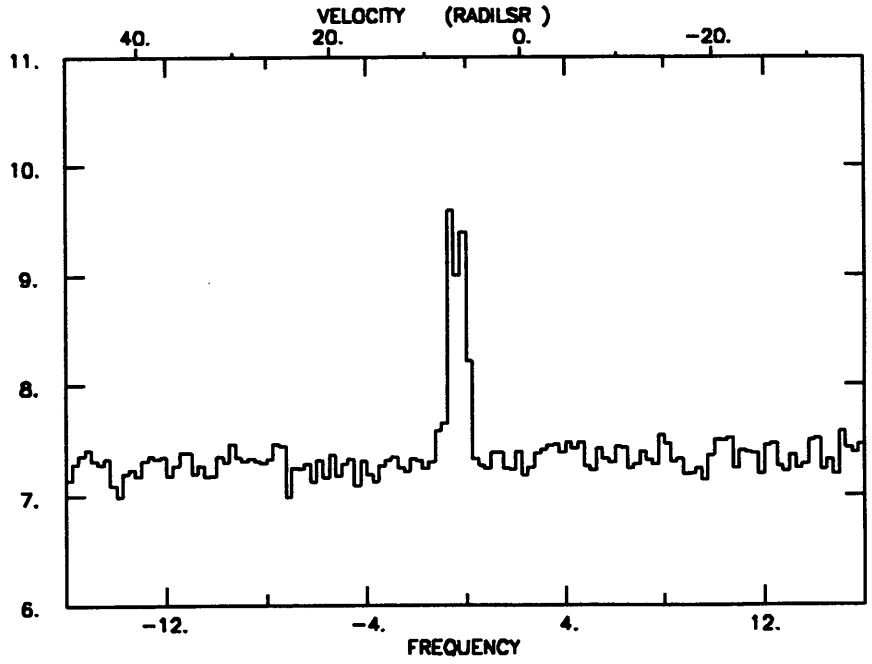


replace xx

uses bad channels identified in last use of badch

Channel numbers may be entered manually with the command **badpt**. See also, **spike**.

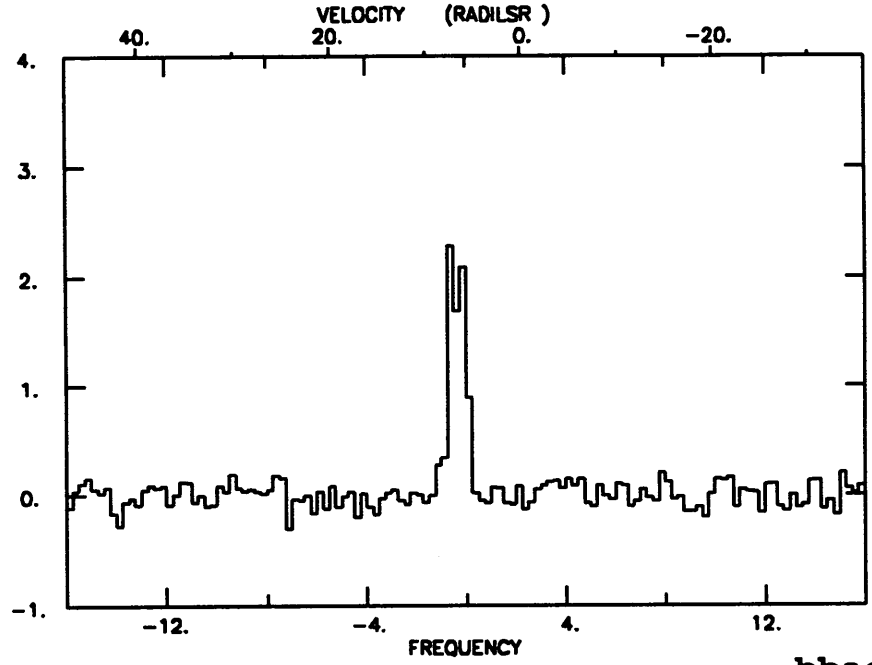
Fitting Baselines



TMC1 19.01 INT= 00:04: 0 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 448
FREQ=115271.20 SYN=1.95990626 VEL= 5.8 DV= -0.65 FR= 250 SB=2

19f
halves
badch

bset
nfit
rms
bbase
ebase



TMC1 19.01 INT= 00:04: 0 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 448
FREQ=115271.20 SYN=1.95990626 VEL= 5.8 DV= -0.65 FR= 250 SB=2

bset
answer question and follow mousing instructions

nfit=1
order of polynomial to be fit

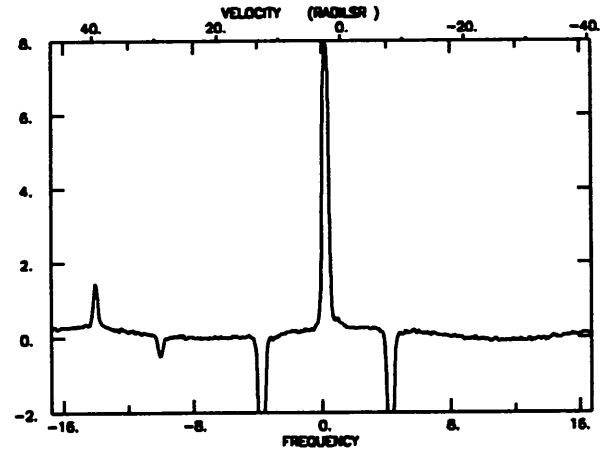
baseline xx

LineF>**rms**

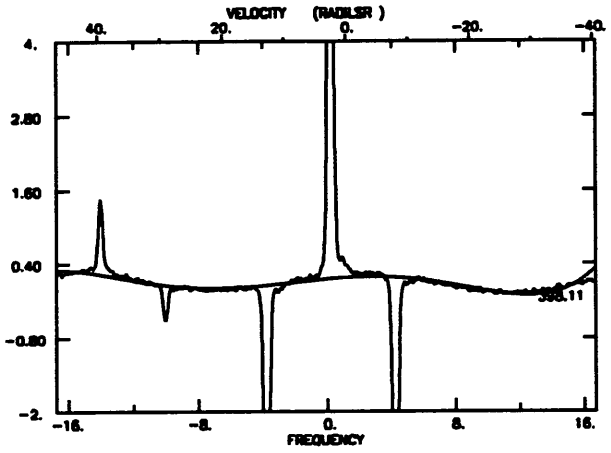
RMS VALUE 0.1099

bbase and **ebase** can also be used to set the first and last nn channels as the baseline regions, e.g.
bbase=30
ebase=30

Fun With Baselines



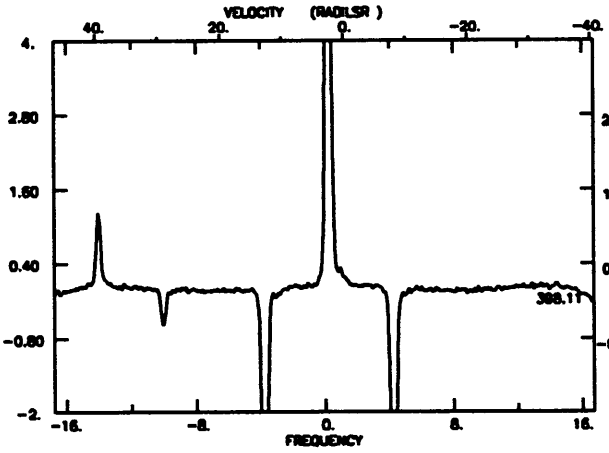
L134 47 SCANS: 398.11- 444.11 INT= 04:38:44 DATE: 19 JAN 95
1950RADC=15:51: 0.0 -04:26:57 (15:51: 0.0 -04:26:57) CAL= 400.0 TS= 518
FREQ=115271.20 SYN=1.99447944 VEL= 3.0 DV= -0.13 FR= 49 SB=2



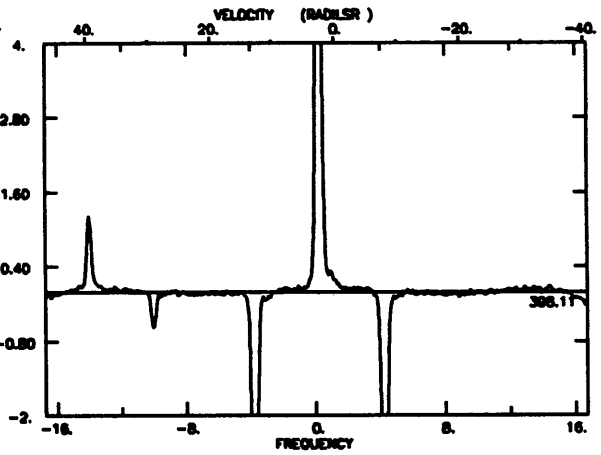

```
slabel=2
-2 4 yrange
nfit=5
bset
```

bshape fits a baseline
bshow plots the fitted baseline on the spectrum

baseline xx



zline=1 xx

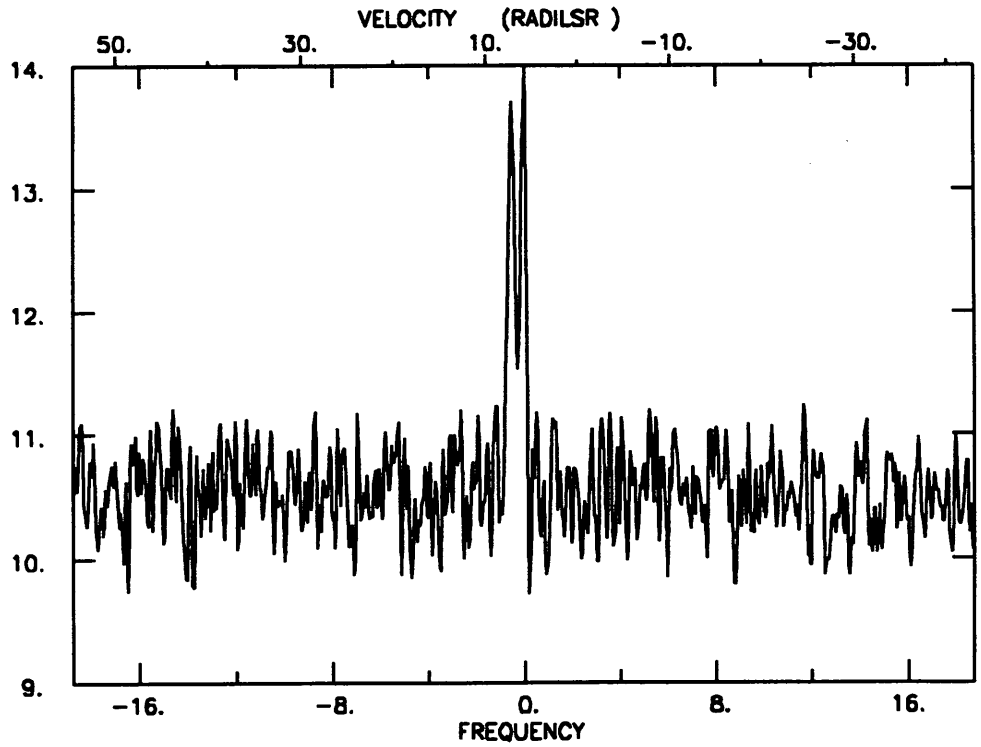


bshow.1 2/13/95

this page dedicated to Paul Hart

bshape bshow destroy baseline

Hybrid Correlator Data 1



TMC1 19.11 INT= 00:01:58 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 461
FREQ=115271.20 SYN=1.95990626 VEL= 5.8 DV= -0.13 FR= 49 SB=2

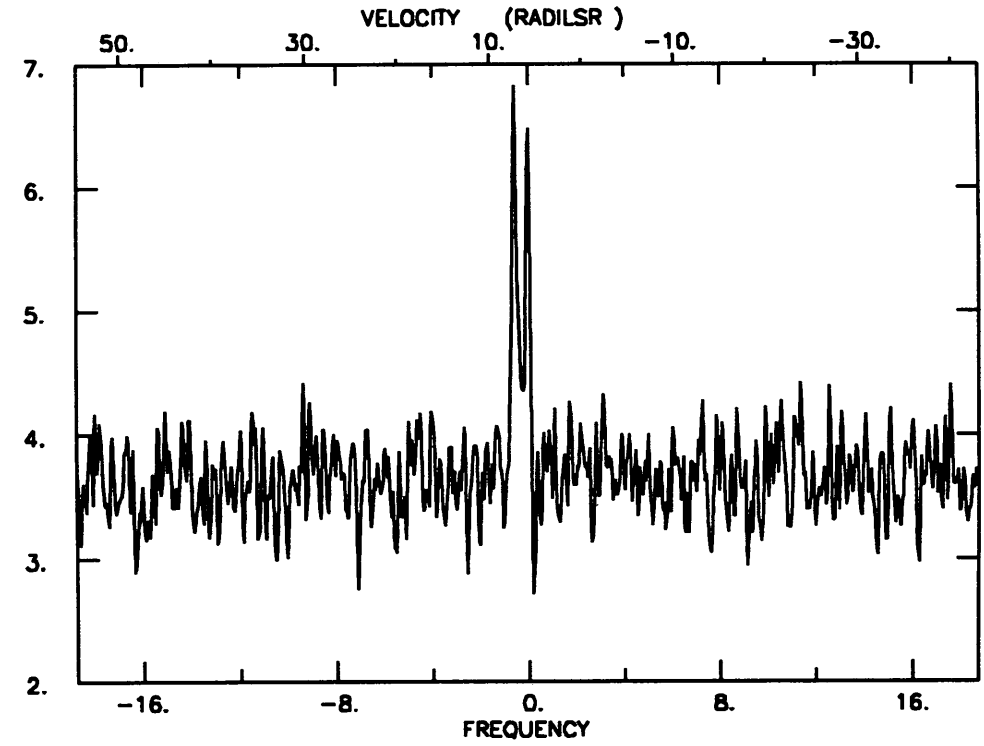
LineF>hcdata
LineH> **19f**

notice that the prompt

is now LineH nstead of LineF

first receiver

data



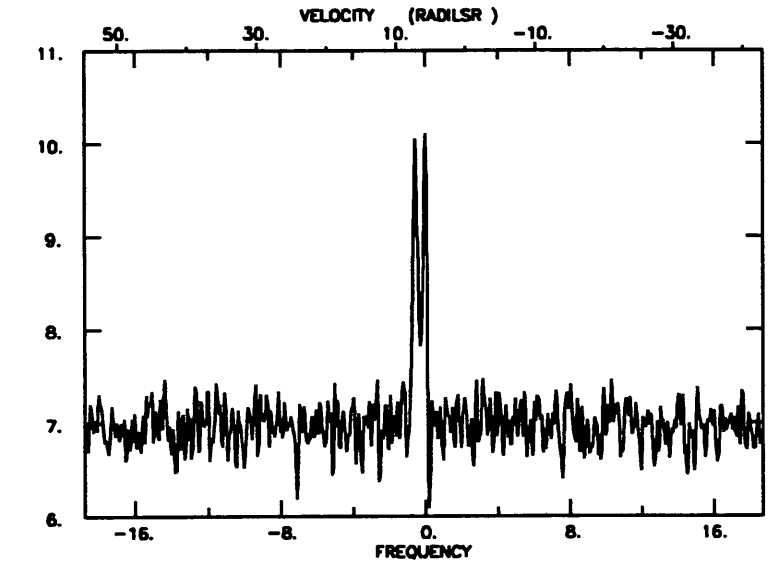
TMC1 19.12 INT= 00:01:58 DATE: 07 OCT 94

19s

second receiver

1 in the tenths digit of the scan # means hc data

Hybrid Correlator Data 2



TMC1 2 SCANS: 19.11- 19.12 INT= 00:03:56 DATE: 07 OCT 94
1950RADC=04:38:38.0 25:35:45 (04:38:38.0 25:35:45) CAL= 400.0 TS= 453
FREQ=115271.20 SYN=1.95990626 VEL= 5.8 DV= -0.13 FR= 49 SB=2

hcdata
empty
19 a
cb

bdata

fbdata
returns to
filter bank
data

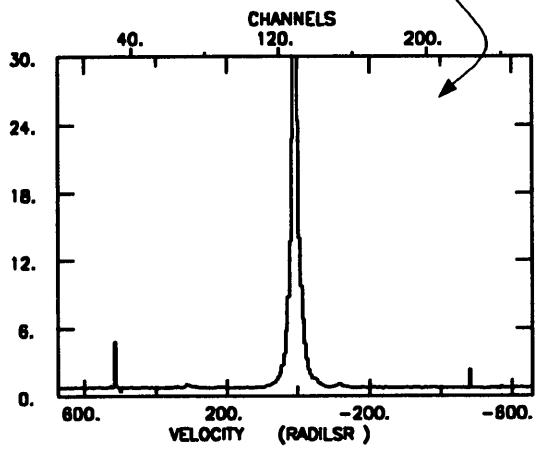
avg_scans_ser_hc.

cstack c1 c2 cb

empty empties "the stack"

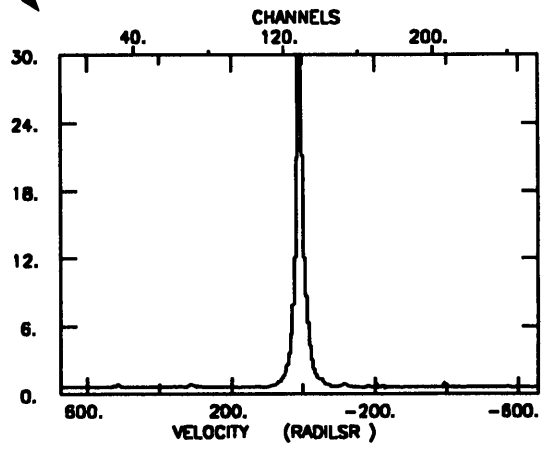
373 375 add puts into the stack a continuous sequence of scan numbers

c1 combines first filter bank: .01's



ORION A 3 SCANS: 373.01- 375.01 INT= 00:18: 0 DATE: 19 JAN 95
1950RADC=05:32:47.0 -05:24:21 (05:32:47.0 -05:24:21) CAL= 400.0 TS= 428
FREQ=115271.20 SYN=1.99394484 VEL= 9.0 DV= -5.20 FR=2000 SB=2

c2 combines second filter bank: .03's

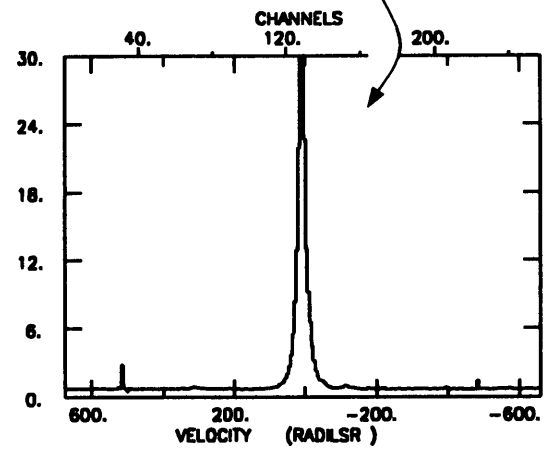


ORION A 3 SCANS: 373.03- 375.03 INT= 00:18: 0 DATE: 19 JAN 95
1950RADC=05:32:47.0 -05:24:21 (05:32:47.0 -05:24:21) CAL= 400.0 TS= 414
FREQ=115271.20 SYN=1.99394484 VEL= 9.0 DV= -5.20 FR=2000 SB=2

a (not pictured) adds a single scan # to the stack

delete (not pictured) removes a single scan # from the stack

cb combines .01's and .03's



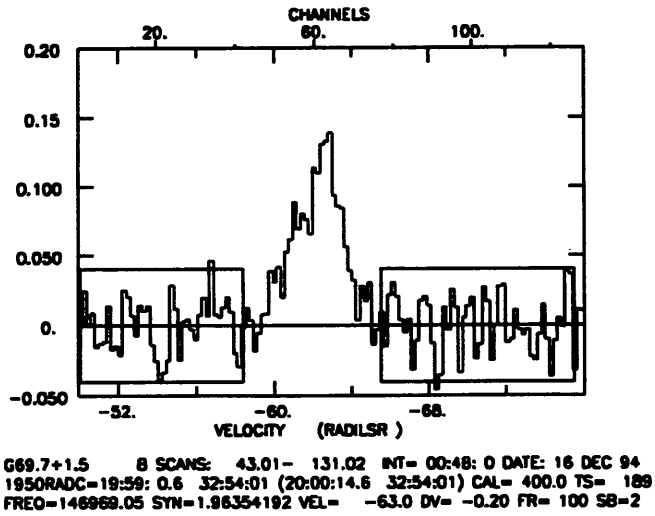
ORION A 6 SCANS: 373.01- 375.03 INT= 00:36: 0 DATE: 19 JAN 95
1950RADC=05:32:47.0 -05:24:21 (05:32:47.0 -05:24:21) CAL= 400.0 TS= 421
FREQ=115271.20 SYN=1.99394484 VEL= 9.0 DV= -5.20 FR=2000 SB=2

c1, c2 and cb ignore subscan numbers

Averaging hybrid correlator scans is done exactly the same way as with fb series data. Use the same stack. Just type `hcdata` and `c1`, `c2`, for 1st and 2nd receiver respectively, or `cb` for both, as desired.

cstack (not pictured) *does not ignore subscan numbers.* Therefore, this command is useful if you don't want to include one or other reciver/filter bank, here and there in a set of scans. Just be sure the stack contains exactly the (sub)scans you want averaged, and cstack will average only those. For example, if you have 1.01, 2.03 in your stack, **c1** will average 1.01 and 2.01 even though the latter is not in the stack. Similarly, **c2** will average 1.03 and 2.03, and **cb** will average all four scans, 1.01, 1.03, 2.01, and 2.03. **cstack** will average only 1.01 and 2.03.

Averaging Scans 2: Parallel Data



`empty` empties "the stack"

`43 a` adds a single scan to the stack

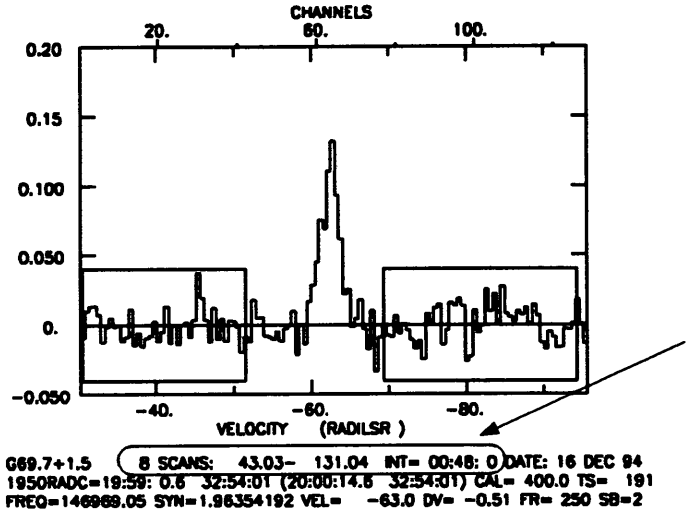
`129 131 add` adds a continuous sequence of scans to stack

`charsize(14)` makes 14 point characters

`c1` combines first filter bank: .01 and .02's

`halves bset`

`baseline xx`



`c2` combines second filter bank: .03 and .04's

`halves baseline`

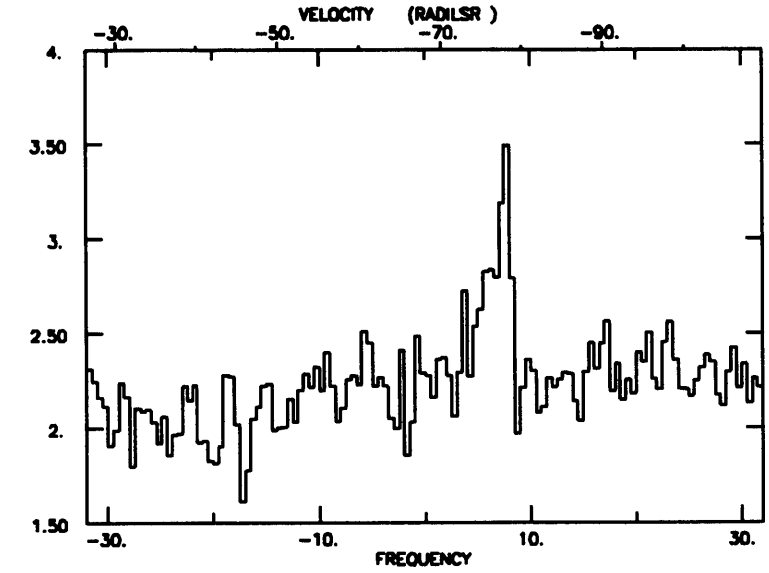
`xx`

(4 scans) x (2 receivers)
= 8 scans
8 scans x 6 minutes/scan
= 48 minutes of integration

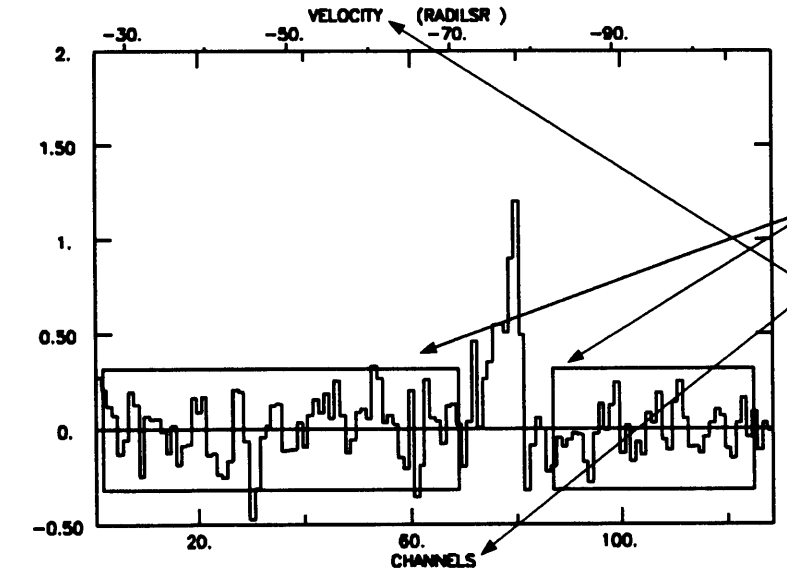
c1
c2
cb

Marking Baseline Regions, and Zero Line, Changing Axes Units

bset,
bmark,
zline,
cv, vc,
cf, fc,
vf, fv,
rms, xx



GBIGA3 198.01 INT= 00:01:40 DATE: 01 DEC 93
1950RADC=20:15:18.7 38:12:32 (20:15:43.0 38:43:20) CAL= 400.0 TS= 479
FREQ=230537.99 SYN=1.90659548 VEL= -68.0 DV= -0.65 FR= 500 SB=2



bset

baseline

bmark=1

cv (also try, vc, fv, fc, cc, vv, etc.)

zline=1

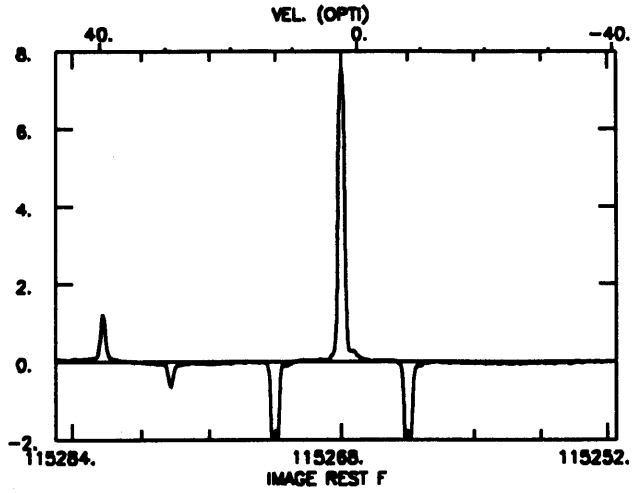
xx (replots)

rms

RMS VALUE 0.1535

GBIGA3 198.01 INT= 00:01:40 DATE: 01 DEC 93
1950RADC=20:15:18.7 38:12:32 (20:15:43.0 38:43:20) CAL= 400.0 TS= 479
FREQ=230537.99 SYN=1.90659548 VEL= -68.0 DV= -0.65 FR= 500 SB=2

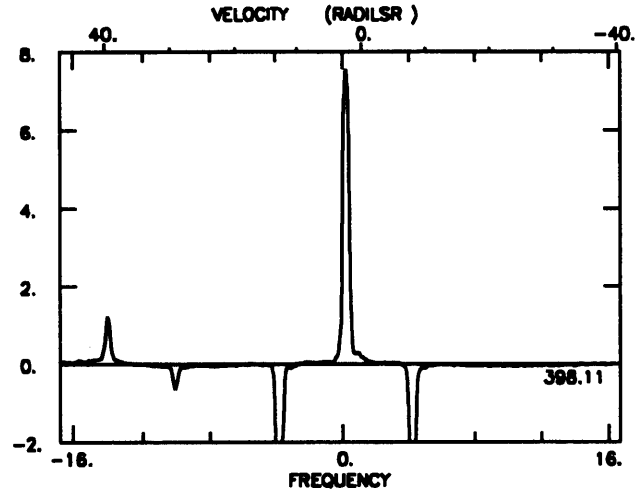
More Elaborate Axes Units



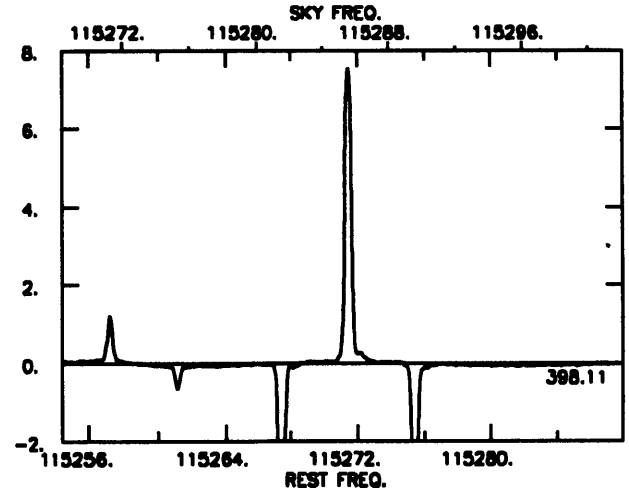
L134 94 SCANS: 398.11- 444.12 INT= 09:17:28 DATE: 19 JAN 95
1950RADC=15:51: 0.0 -04:26:57 (15:51: 0.0 -04:26:57) CAL= 400.0 TS= 488
FREQ=115271.20 SYN=1.99447944 VEL= 3.0 DV= -0.13 FR= 49 SB=2

saxis (*lower units, upper units*)

saxis (freqirst, velopti)
(rest frequency in image sideband, optical velocity)



saxis (freqoff, velhead)
(frequency offset from center, velocity as defined in header)



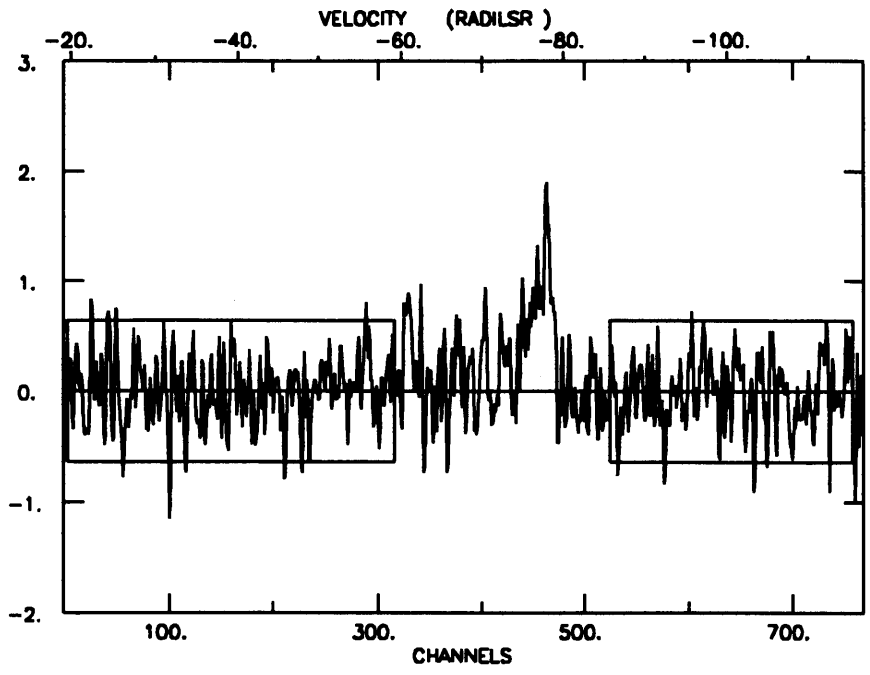
saxis (freqrst, freqsky)
(rest frequency, sky frequency)

other choices are:

vellinr assuming channels equally spaced in velocity
velradi velocity using radio definition of doppler shifts
velrelt relativistic velocities
freqisky (my favorite) sky frequency in image sideband
channel

any combination can be used. UC 6.2

Changing Axes Scales



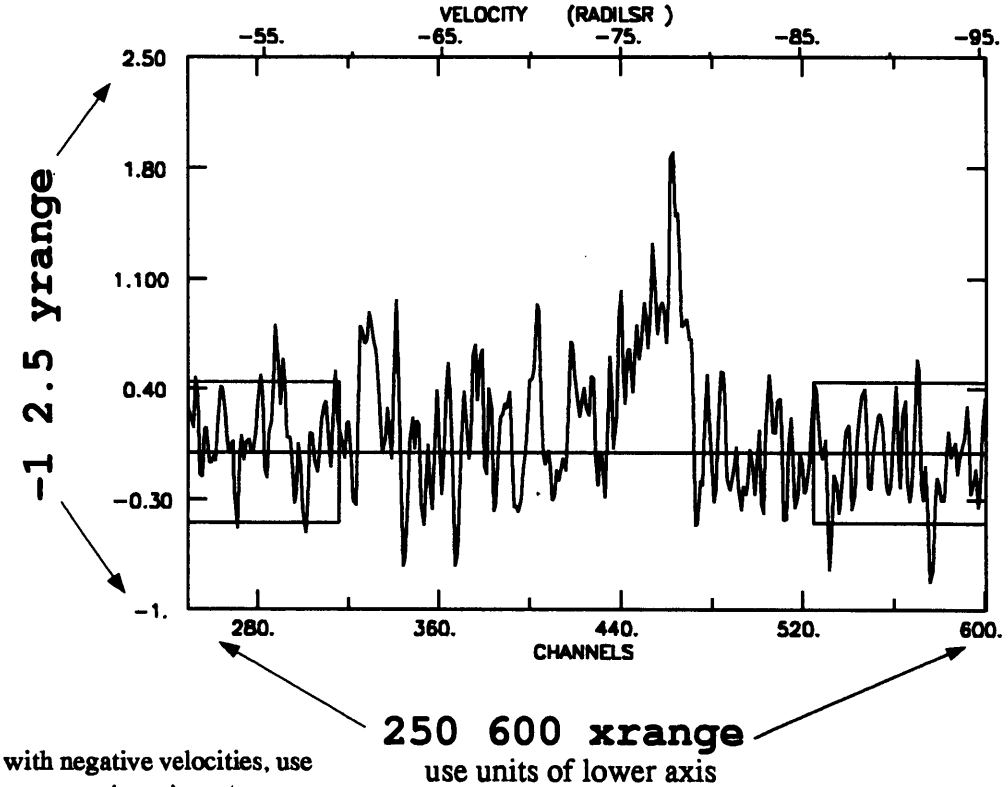
GBIGA3 198.11 INT= 00:00:49 DATE: 01 DEC 93
1950RADC=20:15:18.7 38:12:32 (20:15:43.0 38:43:20) CAL= 400.0 TS= 460
FREQ=230537.99 SYN=1.90659548 VEL= -68.0 DV= -0.13 FR= 98 SB=2

to constrain axis scale:
xrange
yrange

holdy maintains current scale on future plots

to release, use:
freey
freex

can also use
bdrop=nn
edrop=mm
to drop first nn channels and last mm channels

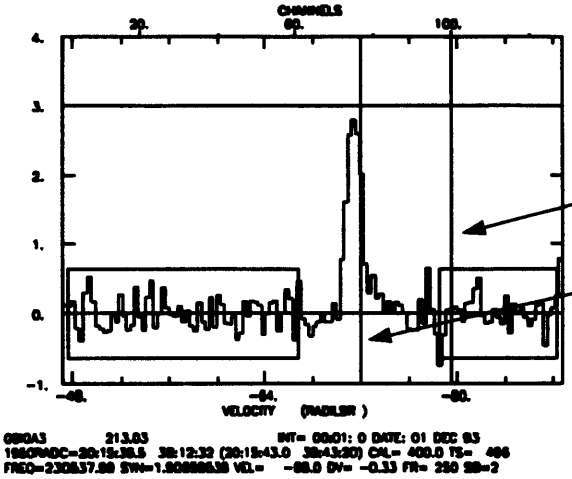


with negative velocities, use
`xrange(x,y)` syntax

xyrange.2

xrange yrange holdy freey freex axes scales

Plot Symbols, Label Sizes
Marking Temperature Velocity, Channel,

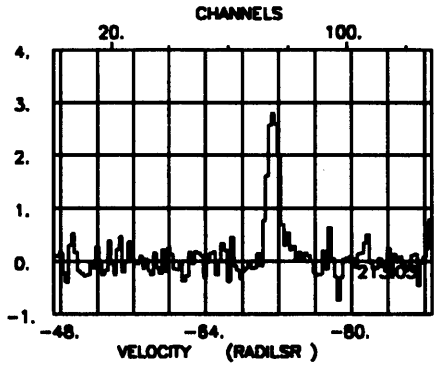


tmark=3

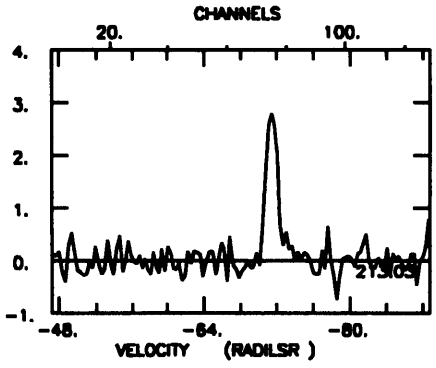
cmark=100

vmark=-72
(**fmark** can also be used)

xx
(zline=1 bmark=1)

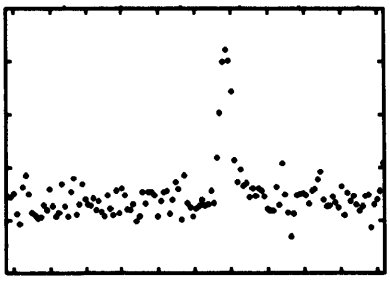


tmark=99 removes mark
cmark=300 removes mark
vmark=0 removes mark
charsize(20)
slabel=2
xx
fullgrid (no xx for fullgrid)



notice how fullgrid goes away on next plot

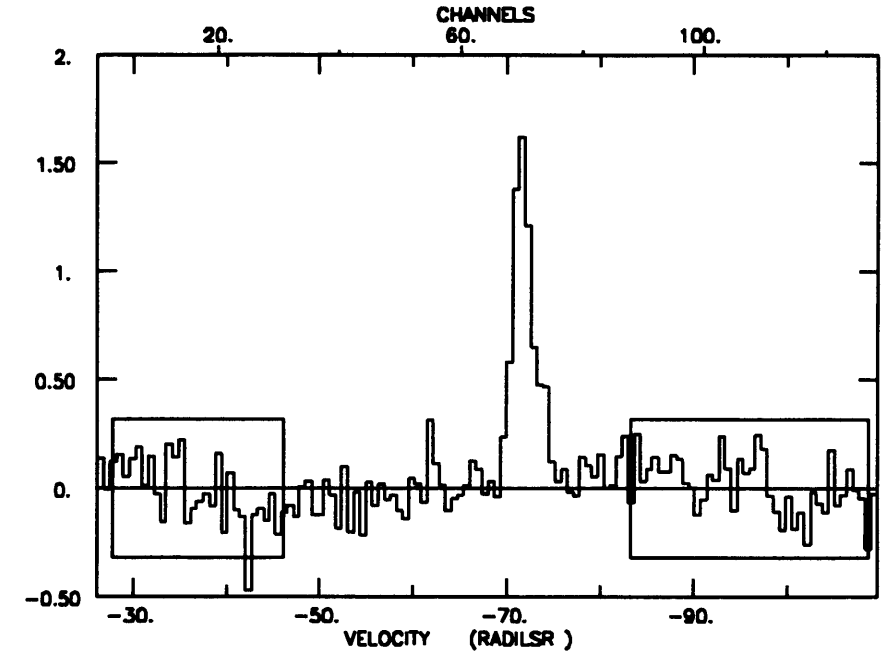
line (singular)
histogram is the default
xx



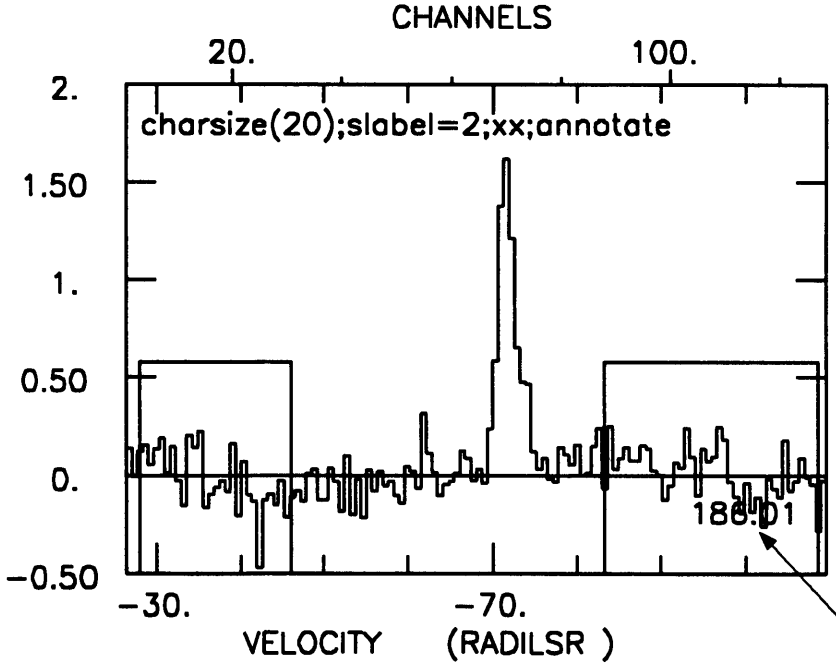
slabel=0
points
xx

slabel=1
returns to default
(not shown)

Annotating Display, Spectrum Label Variations, Typesize, and Line Parameters



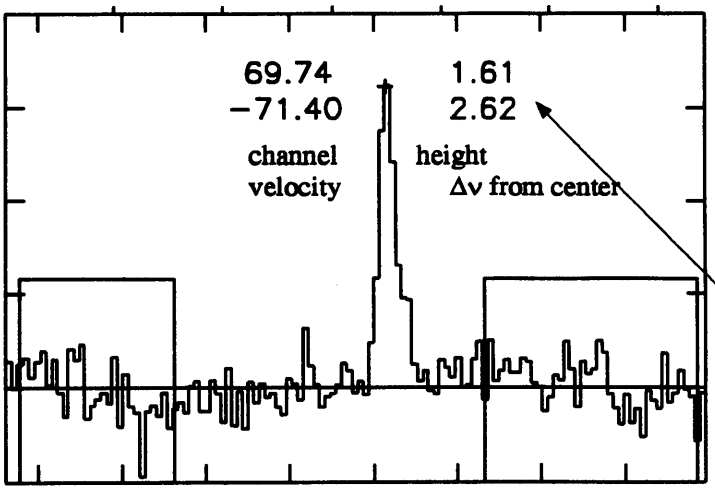
default labeling is like this,
slabel=1



charsize(20)
slabel=2
xx
annotate
Follow instructions for entering string and using cursor to indicate location of beginning of string. Use "'s (quotes) if string contains spaces.

charsize(default)
returns to default

notice the scan number (put there by slabel=2) hiding here in the noise



`charsize(20)`

`slabel=0`
no axis labels at all
`slabel=1` is default

`xx`

`crosshair`
follow mousing instructions

```
LineF>peak
Hght: 1.61985   ; Cntr: -71.5740   (Channl:  70); HW: 3.25099
LineF>rms
RMS VALUE  0.1441
```

You can change what is printed by crosshair. In the example above **`crossflg=11111`**, which is the default.

Parameters returned by **`crosshair`** are set by **`crossflg`**=ctfvm,
where ctfvm are channel, temperature, velocity, frequency offset, and zvalue for 2-D plots. A given digit set to 1 will print the value, if set to 0 that parameter will not be printed. To print only Δv, for example, set crossflg=00010. (The fifth digit is ignored for regular spectra.)

Area under a curve.

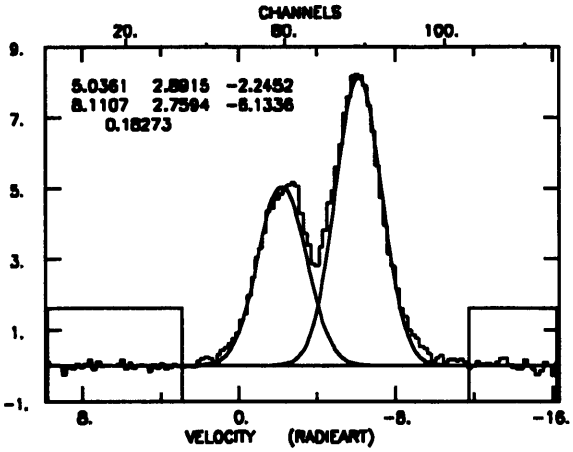
`moment` (not picutred) calculates the value of the 0th and 1st moments, i.e. area under the curve, and maximum value, for the region in Array(0) (what is plotted) defined by **`bmoment`** and **`emoment`**, and saves the result in the array **`size`**.

So, to find the area under the curve:

```
bmoment=left most channel in the line
emoment=right most channel
moment
print size
```

area under curve (units of y-axis times lower axis), intensity-weighted mean x-value (lower axis units)

So, to get Kkm/s, make sure the lower axis is velocity. (e. g. vc, vf, etc.)



gset

follow mousing instructions for setting left and right half power points and peak of gaussians to be fit

gauss

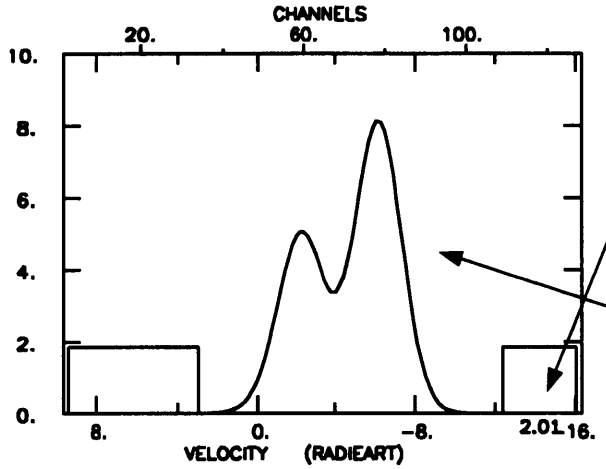
does the math

gparts

fits the gaussians and draws the result on top of the existing plot, fit parameters are shown:

height1 fwhm1 center1
height2 fwhm2 center2
fit quality
units of lower axis

fit quality = sqrt [(chi squared) / (#channels - #free parameters)]



slabel=2

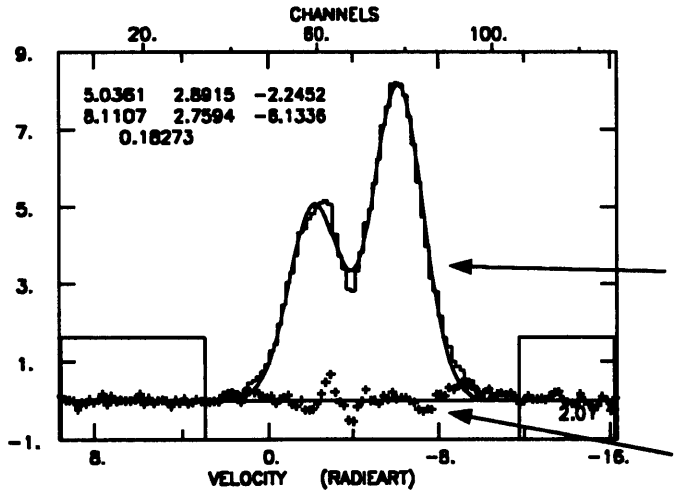
2.01 is the scan number, no other header information

charsize(16)

makes 16 point characters, works for any labeling

gmodel line xx

gmodel puts the sum of the gaussians into the plotting array
`histogram` is the default



2f halves replace baseline
slabel=2 xx

after gmodel, the spectrum must be recalled

gset
gauss
gdisplay plots sum of fit guaussians
residual calculates residuals
rpoints tells reshow to use points, `rline` and `rhistogram` function as the name implies
reshow plots residuals on top of existing plot, using same scaling; r in rpoints is for r in reshow

Tile Maps

follow these steps to make spectra maps like those shown on the following pages

batch maps.plib this just loads the procedures so that you can use them
batch tile.plib run these batch files only once per session

empty empty stack
x y **add** add map scans into the stack
z **a** for filter bank, explicity enter non-integer scan # (.01 or .02)
s t **add** for hc, enter both .11 and .13 scans and prcstak adds them together
as needed

tell stack to see what scans are in stack

prcstk(.001,.001)

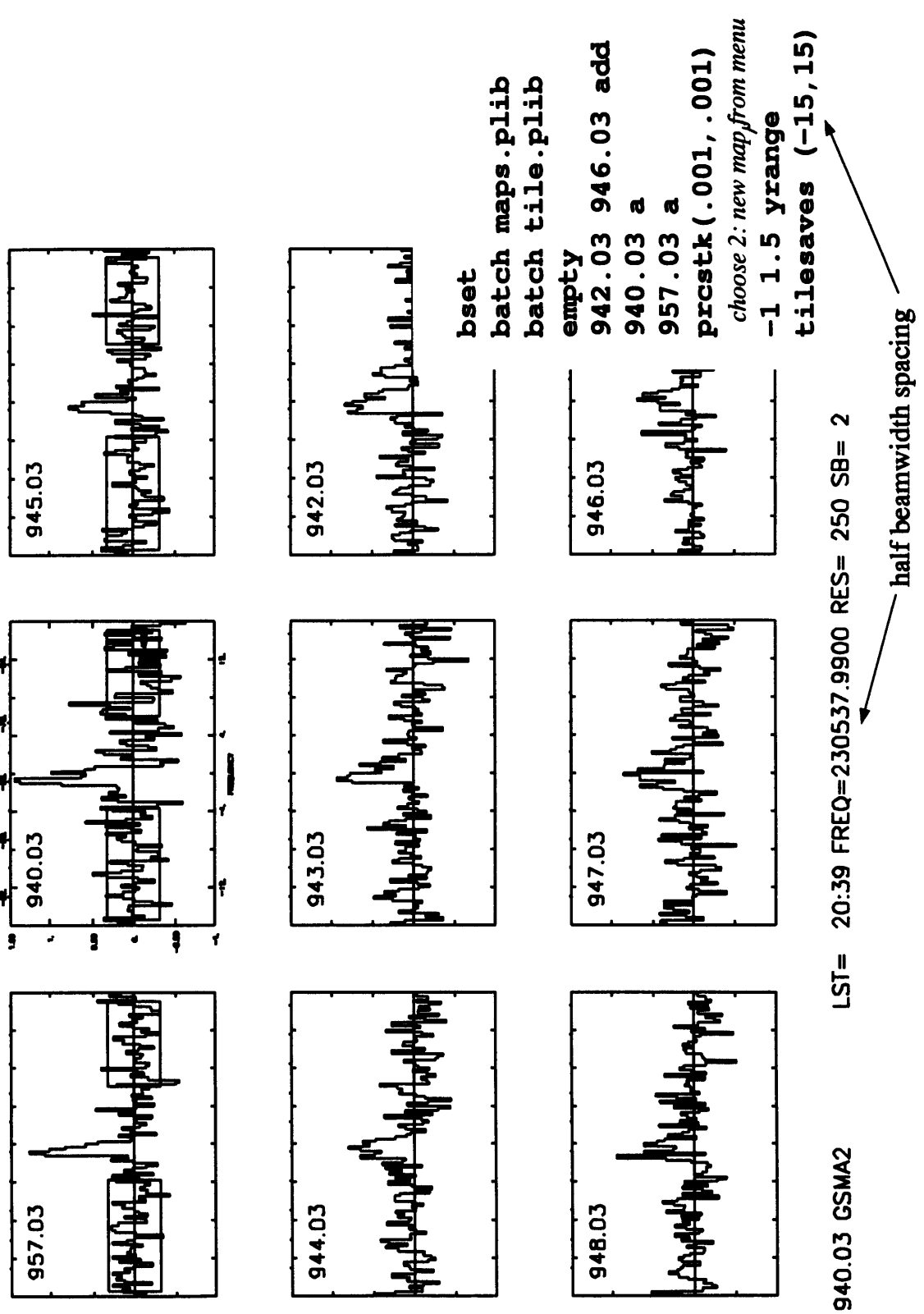
- When starting a map, choose *2, new map*, from menu presented after typing prcstk.
- To extend the map, add additional scans to the stack. Then run prcstak and choose *0, append*, from the menu.
- prcstk averages together scans at same position. In ( ) are pointing tolerances in ". Scans taken at positions within this distance are taken to be at the same position and averaged.
- This procedure does baselining as default, so be sure baseline regions are set properly before starting.
- prcstk can be edited for more exotic processing (like bad channel removal, etc.).

issue aesthetic commands as desired:

bdrop= Tilesaves obeys whatever aesthetics commands have
edrop= been issued. The spectra will be small, so it is helpful
a b **yrange** to dictate the axes scales in case you cannot read
c d **xrange** them once the plot is finished.

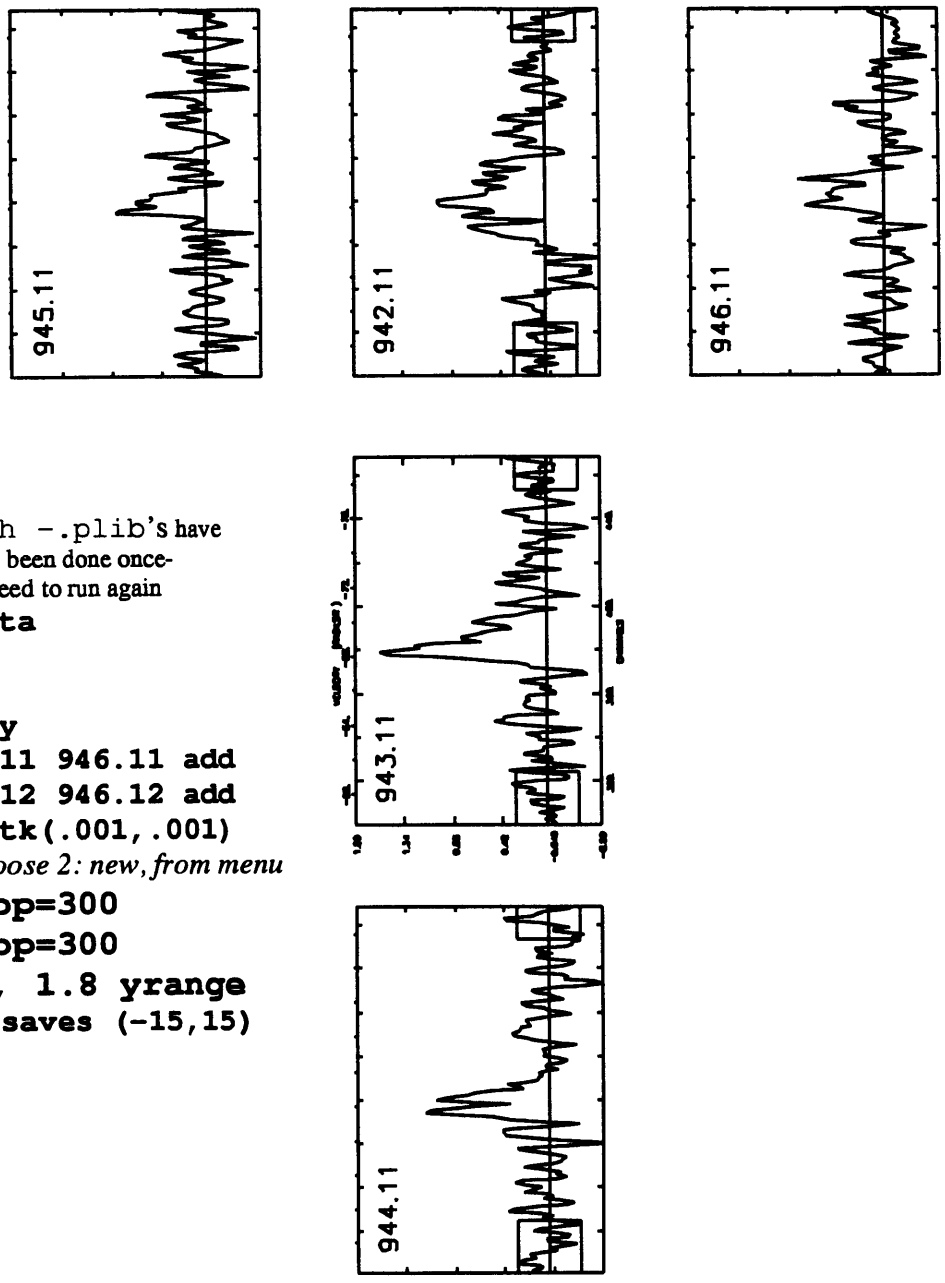
tilesaves(-30,30)
in parenthesis is spacing of data in arc seconds (30" in this case);
[-] yields RA-like x-axis
this is the procedure that contains the command that writes the map to the save area defined above (`filename`)
If you get unexpected results (such as overwriting some spectra) when the plot is made, run prcstk again and choose new map from the menu so it starts over.

Tile Map Example 1



ti_map_1.3

Tile Map Example #2

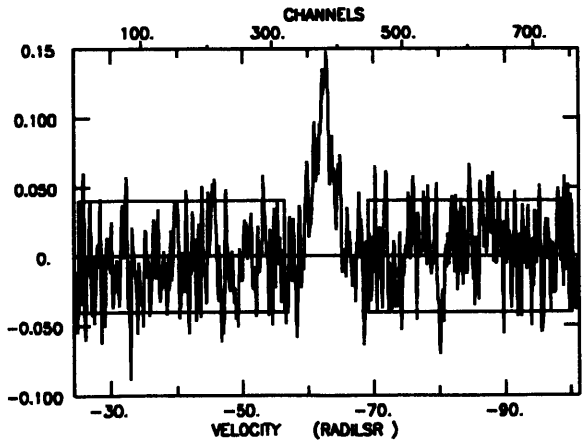


batch -.plib's have already been done once- don't need to run again

hcdata
942f
bset
empty
942.11 946.11 add
942.12 946.12 add
prcstk(.001,.001)
choose 2: new, from menu
bdrop=300
edrop=300
-.5, 1.8 yrange
tilesaves (-15,15)

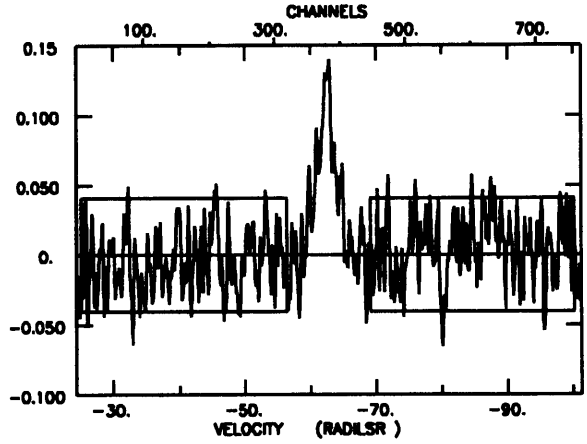
In the tilesaves map, only one scan number is printed for each position, no matter how many scans were added to get that spectrum. If multiple pointings are taken at any position, put all scans into stack, and prcstk does the sorting properly. As prcstk sorts the scans in the stack, it prints messages indicating which scans are added together. In this case, at each position, a .12 scan has been added to the .11 scan (with the same integer part) to get the final spectrum. In parallel mode, put .01's and .02's or .03's and .04's into stack. In series, put in .01's and/or .03's as appropriate.

Smoothing 1



G89.7+1.5 6 SCANS: 129.11- 131.12 INT= 00:35:31 DATE: 17 DEC 94
1950RADC=19:59: 0.6 32:54:01 (19:59: 3.0 32:54:01) CAL= 400.0 TS= 193
FREQ=146969.05 SYN=1.96354334 VEL= -63.0 DV= -0.10 FR= 49 SB=2

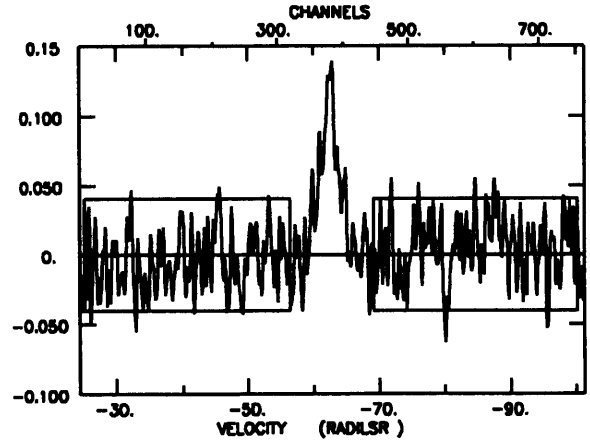
cb
baseline xx



G89.7+1.5 6 SCANS: 129.11- 131.12 INT= 00:35:31 DATE: 17 DEC 94
1950RADC=19:59: 0.6 32:54:01 (19:59: 3.0 32:54:01) CAL= 400.0 TS= 193
FREQ=146969.05 SYN=1.96354334 VEL= -63.0 DV= -0.10 FR= 49 SB=2

hanning xx

no parameters to set,
UC sect. 8.2



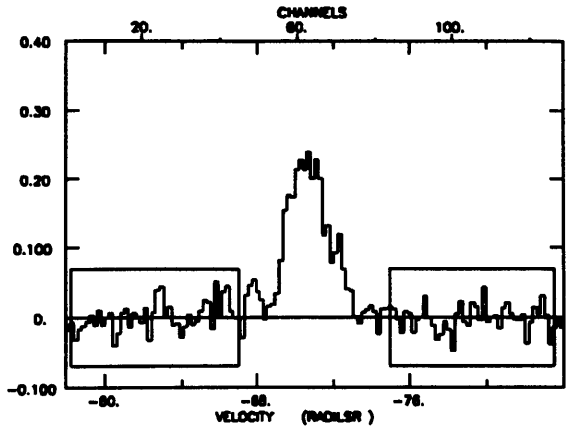
G89.7+1.5 6 SCANS: 129.11- 131.12 INT= 00:35:31 DATE: 17 DEC 94
1950RADC=19:59: 0.6 32:54:01 (19:59: 3.0 32:54:01) CAL= 400.0 TS= 193
FREQ=146969.05 SYN=1.96354334 VEL= -63.0 DV= -0.10 FR= 49 SB=2

cb baseline
nbox=3
boxcar xx

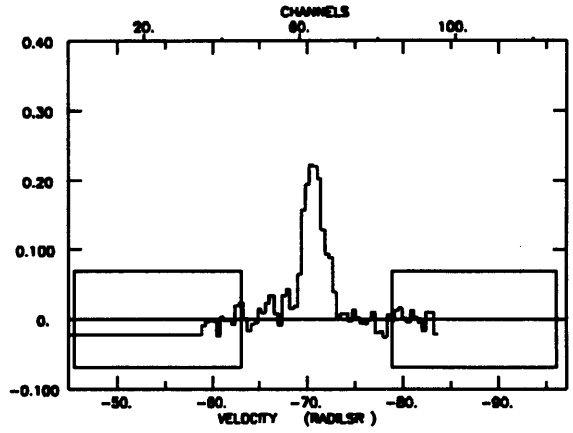
UC sect. 8.1

smothing.3 2/14.95

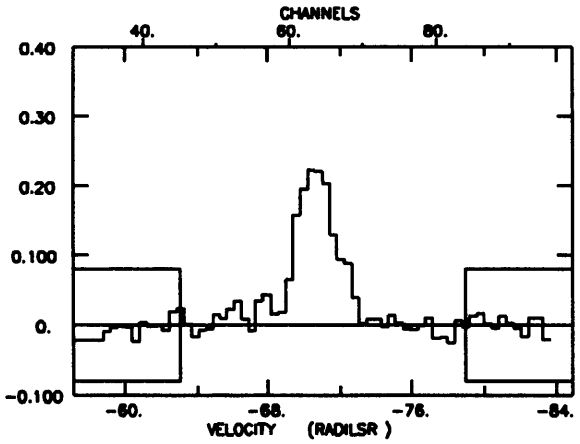
hanning boxcar nbox



G89.9+1.3 10 SCANS: 133.01- 137.02 INT= 01:00: 0 DATE: 14 JAN 95
1950RADC=20:59:55.0 48:43:00 (20:59:55.0 48:42:30) CAL= 400.0 TS= 216
FREQ=146969.05 SYN=1.96359486 VEL= -71.0 DV= -0.20 FR= 100 SB=2



G89.9+1.3 10 SCANS: 133.01- 137.02 INT= 01:00: 0 DATE: 14 JAN 95
1950RADC=20:59:55.0 48:43:00 (20:59:55.0 48:42:30) CAL= 400.0 TS= 216
FREQ=146969.05 SYN=1.96359486 VEL= -71.0 DV= -0.41 FR= 200 SB=2



G89.9+1.3 10 SCANS: 133.01- 137.02 INT= 01:00: 0 DATE: 14 JAN 95
1950RADC=20:59:55.0 48:43:00 (20:59:55.0 48:42:30) CAL= 400.0 TS= 216
FREQ=146969.05 SYN=1.96359486 VEL= -71.0 DV= -0.41 FR= 200 SB=2

`newres=2`

sets the number of channels to smooth by;
must be an integer > 1

`chngres xx`

gaussian smooths and reduces total number of data channels by `newres`.

T=0 channels are appended so that the smoothed spectrum can be averaged with other spectra of appropriate resolution. For example, with `newres=2.5`, the smoothed spectrum could be added to data from the 250kHz filter bank.
more information in UC sect. 8.3

`bdrop=30`
`edrop=30`
`xx`

for more smooting options see UC sect. 8.4 & 8.5,

Chapter 4
Procedures

A procedure is a program composed of unipops commands. There are several built-in procedures and users can write their own procedures. Procedures can contain loops, pass parameters and return values. Comments may be entered on lines beginning with a #. Basic information about editing procedures is provided below, followed by a few annotated examples. The power of procedures is too varied to be thoroughly introduced here. Users wanting to write anything more than simple procedures are directed to *Unipops Cookbook*. Keep in mind that the beginnings of Pops are some 20 years old or more, thus, its function and structure may seem archaic, cumbersome, or incomprehensible. (Perhaps the apparent clarity of Pops is a function of the age of the user.)

`help proc` lists all procedures

`explain` *`procedure-name`* (or *`commandname`*) spawns popup window with VMS style help. This is a more detailed description than is given with help. The increased detail and sophisitication may or may not be useful. Remember, `help` doesn't work for procedures.

`edit` *`filename.prc`* spawns pop-up window with chosen editor on *filename*. The .prc extension indicates that it is a procedure and thus must be used.

Type your procedure into this file, then install it into unipops with the `batch` command. This is the "reccommended" way to enter procedures. This method is the way to edit directly to disk from within LINE. Outside of LINE, you can edit files in the usual way.

`batch` *`filename.prc`* use this command from within line to load the procedure named *filename.prc*

`prcdir` used from within line, spawns a window in which is listed the names of user defined procedures, i.e. those procedures that you have written directly to disk

`type` *`filename.prc`* lists the contents of the file filename.prc; if no filename is given, then unipops will prompt for a file name

`procedure` *`procedure-name`* gives : prompt for entering a procedure line-by-line within unipops

This is the easiest way to enter short procedures.

Type `finish` on last line to end the procedure.

A procedure entered in this way won't be written to disk as a file with a .prc extension and such a procedure cannot be edited except by popsedit as described below. If your procedure is going to be long and you want to make sure it is saved for posterity, don't enter it this way; do it by edit as discussed above.

`list` *`procedurename`* lists, with line numbers, the commands in *procedurename*

`popsedit` *`procedurename line#`* produces : prompt at which you can retype that line

Addtional lines that will be placed between *line#* and *line#+1*

Non-integer line#'s can be used, e. g. `popsedit kathy 2.5` will put a line or lines between existing lines 2 and 3.

`endedit` to finish editing

to remove a line, type `endedit` at the first : prompt

`scratch` *`procedurename`* deletes user-defined procedure named `procedurename`

use `compress to` recover disk space

```
LineF>procedure ba
:# this procedure saves the user the trouble of typing all the commands on the next line
:baseline xx rms
:#fits and removes baseline, replots, prints rms
:# finish indicates the last line of the procedure
:finish
```

```
    SPACE USED/AVAILABLE FOR:
   PROGRAMS  VARIABLES   SOURCE
 13671/32766 2847/32766 10485/32766
 (  42%       9%        32%)
```

(it automatically prints out the above information when you finish entering your procedure)
now make sure that you entered what you thought you did

```
LineF>list ba
  1 procedure ba
  2 # this procedure save the user the trouble of typing all the commands on the next line
  3 baseline xx rms
  4 #fits and removes baseline, replots, prints rms
  5 # finish indicates the last line of  the procedure
  6 finish
LineF>
```

Example 2: **A Predefined Procedure, f, Annotated**

Let's try to understand what happens when you type `scan# f`

```
LineF>list f
F -> S1
```

```
LineF>list s1
  1 procedure s1(xscan)
  2 disp12m(xscan, 1)
  3 return
  4 finish
```

```
LineF>list disp12m
  1 procedure disp12m(xscan,fb)
```

it gets xscan from the scan number typed with "f" and fb from the 1 (or 2) passed from s1 (or s2).

```
  2 # Display the data from filter bank fb for scan xscan
  3 global scalar spectype
  4 scalar fscan, nchan
```

Global variables can be accessed from the command line prompt or from other procedures. Local variables can be used only in the procedure in which they are defined. For example, you can type `print spectype` from

the command line prompt and the computer will return either 1, for hcdata or 2 for fbdata. If you ask it the value of nchan, it won't know what you are talking about.

```
5 if spectype = hctype; then
```

Typing hcdata on the command line makes spectype=hctype=1.
Typing fbdata on the command line makes spectype=fbtype=2.

```
6   fscan = newfeed(xscan,(11 + (fb-1)))
```

newfeed adds (11 + (fb-1)) to xscan
line 6 makes a non-integer scan # out of an integer scan #, specifically, either .11 (c1 -> fb=1) or .12 (c2 -> fb=2) [which is a bit of a misnomer in the case of the hc because .11 is really rx1 and .12 is really rx2.]

```
7   get(fscan)
```

Puts data from scan fscan into array 0.

```
8   nchan = 1
```

hcdata has a spectype of 1 and nchan is keeping track of spectype and fbmode. We are in this part of the loop because we have hcdata, so set nchan=1 for later use in deciding to use a regular (rather than side by side) plot of the spectrum.

```
  9 else
```

If not hcdata then it's fbdata, so calculate the subscan number appropriately.

```
10   fscan = newfeed(xscan,(1 + (fb-1)*2))
11   get(fscan)
```

lines 10 and 11 get the .o1 (for f or s1) or .03 (for s or s2) scan

```
12   fbmode(nchan)
```

fbmode 12 queries the filter bank mode, returning nchan=1 for series, or nchan=2 for parallel, nchan is just a dummy variable, you could call it kathy or phil.

```
13 end
```

end of loop which assigns appropriate subscan numbers

now begin plotting
how the data are plotted will depend on whether it was taken in series or parallel

```
14 if nchan = 2; then
```

nchan=2 means parallel mode. If parallel mode put first rx (currently in array 0) in array 3 (line 15)

```
15   copy(0,3)
```

Now make the subscan # refer to second rx. Line 16 adds 1 to the subscan #

```
16   fscan = addfeed(fscan,1)
```

put second rx into array 4 with the command get4. (get puts the data into array 0)

```
17   get4(fscan)
18   cboth
```

`cboth` is the command which produces the side by side display used for parallel data. (Do not confuse `cboth` with `cb` which combines both polarizations of scans in the stack.) cboth plots on the left the data in array 3 and, on the right, the data in array 4 is plotted. The variable names for these data arrays are actually d0, d1, d2, d3, d4. d0 is the plotting array and d1 and d2 are reserved for the gains and other specific verbs.

```
19 else
```

If it isn't parallel data then plotting is simple: just one plot. So, clear the plotting window and plot.

```
20    page; show
21  end
22  return
23  finish
LineF>
```

Example 3: **A User-defined Procedure to Smooth**

```
LineF>batch box2.prc
Batch File: box2.prc
>procedure box2

Procedure already exists...
Do you want to overwrite (y or n)? y

:# Reduce resolution and number of channels by a factor of two.
:# Each channel in a pair is replaced with the average of those two channels
:# Note: This can be adapted to reducing the resolution and number of
:#       channels by any factor
:# declare variables
:scalar i l k x1 x2 av
:for i = 1 to 64
:# step through each fb 2 channels at a time
:   l = 2*i
:   k = 2*i - 1
:# THIS PART OF THE LOOP DEALS WITH THE SIDE BY SIDE DISPLAY
:# d3 is a system defined array that contains the first reciever data
:   x1 = d3(l)
:   x2 = d3(k)
```

```
:# average channels
:    av = (x1 + x2)/2
:# replace values
:    d3(l) = av
:    d3(k) = av
:# d4 is a system defined array that contains the second reciever data
:    x1 = d4(l)
:    x2 = d4(k)
:# average channels
:    av = (x1 + x2)/2
:# replace values
:    d4(l) = av
:    d4(k) = av
:# THIS PART OF THE LOOP DEALS WITH THE "HALVES" DISPLAY
:# d0 is a system defined array that contains the halves data
:    x1 = d0(l)
:    x2 = d0(k)
:# average channels
:    av = (x1 + x2)/2
:# replace values
:    d0(l) = av
:    d0(k) = av
:# finish loop
:end
:# replot
:xx
:return
:finish

      SPACE USED/AVAILABLE FOR:
   PROGRAMS   VARIABLES    SOURCE
 14737/32766  2909/32766 12261/32766
 (  45%         9%        37%)

End of batch file box2.prc
:
```

Example 3 **Averaging Saved Scans**

```
LineF>list avesave
   1  procedure avesave(s1,s2)
   2  #This procedure averages scans that are stored
   3  #in save slots s1 through s2
   4  #declare variables
   5  scalar s1,s2,i
   6  #clear accumlation
   7  #technically sclear turns the accumulator flag off
   8  #accum turns the accumulator flag on
   9  sclear
  10  #loop through save slots
  11  for i = s1 to s2
```

```
12 #identify a save slot
13     nsave = i
14 #get data
15     recall
16 #add it to the already accumulated data
17 #each scan is weighted by the integration
18 #time and inverse system temperature
19 #the sum of the weights is also retained.
20     accum
21 end
22 #finish the average process
23 #ave normalizes by the sum of the
24 #weights and puts the result back
25 #in array 0.  Ave also turns off the
26 #accumulator flag
27 ave
28 #plot contents of array 0
29 #
30 xx
31 return
32 finish
```

Chapter 5 5-1
On the Fly Mapping

otf.5 5/3/95

The observer is vigorously urged to read **"A Guide to Spectral Line On-the-Fly Maping"**, by Phil Jewell. This memo contains important information about, among other things, how to set up your map(s) to acheive the desired noise level. It can be found in anonymous ftp to heineken.tuc.nrao.edu, in the directory observerinfo/postscript, in the file otf_prj.ps. This chapter in the Examplebook is meant as the most rudimentary summary of what is contained in the above memos.

To reduce your data, you will need the Jeff Mangum memo **"How to Process Spectral Line On-The-Fly Data"** which can be found in the same directory as "A Guide..." in the file otfinfo.ps.

1. You tell the telescope operator what map input parameters to use. (Though, in the future, the observer may be abl e to enter them directly.)

`OFF integration time` typically 10 to 15s

`CAL integration time` typically 5s

`Rows per OFF` want 1 to 2 min between OFF's. Time efficiency is maximized when this is an even number.

`rampup distance` *normally 1'* This is extra width added on to both sides of your map to account for the distance it takes for the telescope to accelerate to scanning speed .

`scandist` width of the map

`scanrate` *The observer must calculate the optimum scanrate in arcseconds per second, "/s.* Some of the constraints are 1. make as few maps as possible which means using the slowest scan rates possible but 2. rows should take a minute or less to complete and entire maps should take no more than 30 to 45 minutes to complete. There are several examples in the"Guide to OTF" memo.

`spacing` spacing between rows in ", should be less than Nyquist spacing,
The "Guide" suggests .9(λ/2D) - 2"

`#rows` the number of rows that you want in your map

2. For each map that you make, tell the operator to start a new data file. Due to the amount of data in an OTF map, about 72 Mb/hour, data files with many OTF maps are so large that they are impossible to deal with. However, if your maps are small, then you can allow several maps in the same sdd file. If your source needs several maps, it is convenient to have them in as few files as possible. However, the observer must be mindful of the space on the disk, so as not to suffer the ignominy and ostracism of being the cause of problems associated with insufficient disk space.

Make a note of the filename and the source. The data filename will be sdd.xyz_nnn, where xyz are the observer initials and nnn is the version number (001, 002...). The version number gets incremented each time you start a new file (so "version" is a misnomer, but it's called that for historical reasons.) The gain file name will be gsdd.xyz_nnn. (When you start reducing your data, you will be prompted for these filenames.)

3. Do a regular scan (e.g. APS , PS, FS) before you start your OTF map. Determine the bad channels (with badch, for example) and tell the operator to remove them. Bad channel removal can be done in AIPS, but doing it this way is much easier. You will need other information from this (or these) spectrum later as input to AIPS tasks. This information includes channel numbers for the baseline regions and a channel range for dertermining the integrated intensity. (The `crosshair` command in unipops handy for determining channel numbers, etc.)

4. Make a note of the exact scan numbers in your map. You will need these later when reducing your data in AIPS.

5. You can see a "slice" of your data by following the instructions in the *"Guide"*, which also explains the meaning of the information displayed by the dataserve program on Tecate.

6. Some crisis will happen if AIPS and Unipops (line) try to read the same data file at the same time. If you try it, you'll

Chapter 6

Save Files, Keep Files and Postscript Files

Save files and keep files are files in which you can store scans, usually reduced data. The observer can then use those reduced scans to prepare plots for publication. The use and function of these two types of files are similar, but save files are likely to be more useful than keep files. Save and Keep files are sdd (single dish data) format.

If you want a postscript file of any spectrum that you make with unipops, make a capture file. Capture files don't save the data, just a particular rendering of it. So, capture files serve an entirely different function than save- and keep- files.

Specifying Save and Keep File Names

When you start line, the names of the save and keep files, as well as some other files, are printed out in a table. The *savefile*'s default name is LSAVE, and the *keepfile*'s default name is LKEEP. (You can see these filenames if you do an `ls` in your obs directory.) To change the files that `save` or `keep` writes to, issue the command (see section 5.3 in UC for more information):

`chngfile create 2` *`filename`* for a *keepfile* (2 is code for keepfile)
`chngfile create 3` *`filename`* for a *savefile* (3 is code for savefile).

Listing Scan Numbers in a Save or Keep File

Use the `tell` command. For example:

```
LineF>tell kscans
K   1: 131.11  259.11
LineF>tell sscans
S   1: 1133.01 1133.01  173.01  174.01  175.01  176.01  177.01
S   8: 178.01  179.01  180.01  181.01  182.01  183.01  184.01
```

Save Files

`nsave, save, recall, sprotect, check`

`save` writes the most recently plotted spectrum to the `nsave`'th slot in the *savefile*. (It is most accurate to say that the contents of array 0 are saved to the n'th slot. For example, if you plotted something, then used `get` on another scan, the scan that you '`got`' will be saved, not the most recently plotted spectrum.) When you start line, the parameter `nsave` is set to 1. To save into other than slot 1, type `nsave=m`, where `m` is any number, then `save`. To display the scan in slot 1, type `nsave=1, recall, page show`. In the example above, to see a plot of scan 181.01, type `nsave=11, recall, xx`.

Overwrite protection is set by the paramteter `sprotect`. The default, `sprotect=true`, is to prevent overwriting. To overwrite a slot, type `sprotect=false, nsave=x, save`, where x is number of the slot into which you want to put the spectrum. The currently displayed scan will be saved in slot x. `check` returns the scan number of the spectrum in the current 'save bin'. Having to type `nsave` all the time is tedious but allows more flexibility than is available with *keepfiles*.

Keep Files

`kprotect, keep, kget`

Keepfiles work much the same way as savefiles but the slot number is the same as the scan number . Therefore, no matter how many times you keep a particular scan, only one version can be retrieved. The commands are `kprotect, keep,` and `kget` *`scan#`*, where the *scan#* must have the correct decimal part (.01, .02, .03, .04, .11, .12). So, `keep` is simpler than `save` but less flexible.

Postscript or "Capture" Files

A postscript file (actually, EPS objects) can be made of any spectrum, but it is not simple to go back and forth between sending plots to the printer or disk. To direct output to a file rather than the printer, type the command `setenv popsprinter capture` *before starting line* (i. e. at the **locura** prompt). Then, when you are in **line** and you type `make`, (or `tcopy`, `gcopy`, etc.) a "picture" the spectrum that you have on the screen will be written to a file called *capture.file.xxx*, where xxx is a number. The name of the file will be echoed to the text screen in which you typed `make`. You cannot go back and forth between `make`ing printouts and `make`ing postscript files without exiting and restarting line. Typing `system setenv popsprinter capture` from within line doesn't work. However, you could run line in two different windows or on different terminals.

This is a useful process when you are remote observing because you can ftp the postscript files and print them out where ever you are. It is tedious but reliable.

Chapter 7
Other Useful Commands

`uni2fits` translates your data from single dish data to fits format. simply type uni2fits and you will be queried for filenames, etc. and presto! your data is translated. you can now ftp your data for use with your favorite software!

`eqtogal(`*hhmmss.s,ddmmss*`)galtoeq(`*ddd.ddddd, sdd.ddddd*`)` coordinate transformations (1950)

`system` *unixcommand* shell escape, for example `system ls` lists files in your directory

`page` clears graphics window

`show` plots contents of plotting array (often referred to as array(0))

`xx` same as `page show`

`tcopy` sends to printer last 20 lines of text in the text window

`gcopy` same as make, sends plot to printer

`get scan#` gets the scan but does not plot it, useful with command `header`

`header` prints in text window brief header information for the scan most recent called up with `f, s, get,` etc.

`default` useful with commands like `charsize, bdrop` to return to default setting, e. g.

```
charsize(default)
```

`read` *command* puts you in "input mode" , sometimes this is an easier way to enter parameters

```
>read bdrop
#20
```

`tell` *item* lists contents of the item, e. g. `tell stack` lists the scans in the stack, `tell ondata` lists scan #'s in the on-line data file,. works with `disk, dscans, kscans, sscans, ondata, offdata, gscans` some examples:

```
LineF>tell ondata
O   1: 608.01  608.02  608.03  608.04  609.01  609.02  609.03
O   8: 609.04  610.01  610.02  610.03  610.04  611.01  611.02
O  15: 611.03  611.04  612.01  612.02  612.03  612.04  613.01 etc.
```

`summary` *datafile-type* where datafile-type is `ondata, dscans, offdata, kscans, sscans,` and `gscans` (see section 5.5.1 of UC for complete explanation of these file types). `Summary` gives a summary for each scan in the file. Each line contains (for spectral line data), scan #, receiver #, source name, observing coordinates, frequency resolution, and rest frequency

`setenv DISPLAY hostname:0` If you want to run line on locura while logged in at the bohemia console, type this command with locura as the hostname and you will get a display just as if you were logged into the console of locura. see `xhost`

`xhost +` *hostname* Typed in the console window of your monitor, this allows *hostname* to display things on your monitor. This is automatic for most cases that you will use on the mountain, so you probably don't have to type this

`stty erase ^H` Sometimes the backspace key doesn't work when you are rlogin'd or telnetted. Type this command before entering line and the backspace key will work properly. After "erase", push the space bar and then the backspace key. A ^H will be echo'd and the next time you hit backspace, the cursor should actually move backwards, erasing what was previously typed in that space

`chngver` used to change the online data file. `chngver` prompte the user for FB or HC and lists available version numbers. When you make more than one data file during an observing run, each file gets a new "version" number. Your files have names like sdd_knm.001, where sdd means single dish data (the gsdd file contains the gains), knm

are my initials, and 001 is the version number. When you start a new data file, it will be .002 and so on. When you make on-the-fly maps, usually a new data file is started for each new source.

`chngonline (file_type, version_number)` where filetype is `hctype`, `fbtype` or `contype` (for continuum data) and version_number is as described in `chngver`. `chngonline` and `chngver` perform the same function, though to use `chngonline` you need to know the version number that you want. There are subtlties having to do with changing assumptions about subscan number made by various verbs (see UC section 5.4.2).

`cget` *rx#* gets the current (partially completed) or most recent scan, reciever # is, for parallel data for example, 1,2,3,4,11 or 12, referring to, respectively, first rx-first fb, second rx-first fb, first rx-2nd fb, 2nd rx-2ndfb, 1st rx-hybrid correlator, 2nd rx-hybrid correlator.

To plot on screen, type `xx` or `page show`.

`gget` *scan#* Puts the gains for the specified scan into array (2). For integer scan #'s, the .01 scan is retrieved. The non-integer part may be specified. To plot the gains type

`copy(2,0) page show` This moves the gains from array 2 (where gget puts them) into array 0, the plotting array, clears the screen, and plots them.

`cget feed#` Retrieves the specified feed (e. g. 1, 2, 3, 4, 11, 12) for the current scan and places the header and data into Array 0. use xx to plot

`select` A potentially useful way of selecting scans to put in the stack. Scans can be selected by scan#, mode, source name, LST, UT, coordinates, bandwidth and/or rest frequency. Type `select info` and `explain select` for more information.

`print` prints on the screen the value of named parameter, see example of `print badpt` below

`spike` Sort of like an automatic `badch`. `spike` finds the channel numbers for all data points in Array 0 (the array that is plotted or would be if xx was typed) that deviate more than `cutoff` from zero. The results are stored in badpt array. This is the array used by `replace`.

```
LineF>print badpt
 34.00000     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
LineF>print cutoff
 1.000000
```

Miscellaneous Useful Things to Know

old catalogs can be found in obs/catalogs/xxx where xxx are the observer's initials

lockfiles If certain observer initials start line in a second (or more) window, threatening messages will appear on the screen. It is ok to plunge ahead and run multiple copies of line. Care must be taken when updating the recover file.

Index

Unipops Examplebook for 12 m Spectral Line Users