Manual de observaciones de los telescopios RAEGE en antena única

A. Moreno Signes

Informe Técnico IT-OAN 2015-17

Los desarrollos descritos en este informe técnico han sido cofinanciados por el Programa Operativo Fondo Tecnológico FEDER de la UE bajo el convenio IGN-MINECO denominado “Radiotelescopio de VLBI geodésico y astrométrico para su integración en la red VGOS (VGOSYEBES)”

Revision history

Version	Date	Author	Updates
1.0	05-12-2015	A. Moreno Signes	First version

Índice

1. Introducción

En este manual se resumen todos los procedimientos necesarios para poder realizar las observaciones básicas de caracterización de las antenas de la RAEGE y el análisis y procesado de los datos mediante el paquete de utilidades del IRAM, GILDAS. Se explicarán las funciones principales del software de control, como preparar las macros según qué tipo de observación vayamos a realizar y cómo analizar los datos de las observaciones. Además se añaden algunos apéndices con tablas y ecuaciones que pueden ser útiles durante el proceso de planificación de las observaciones.

2. Control de la antena

En este apartado se resumen algunas de las funciones mas básicas de control de la antena y las mas utilizadas en las observaciones de caracterización que se explicarán posteriormente. Más información sobre el funcionamiento de todas estas funciones se puede encontrar en */home/raegemgr/raege/raegeObserver/src/raegeCommands.py*

2.1. Funciones básicas

- **Control básico**

  Arranque de raege.

```
raege\$ raege
```

  Sale de raege.

```
>bye()
```

  Detiene inmediadamente los motores, argumento por omisión 'm1', las opciones son 'm1' y 'm2'.

```
>stop(motor)
```

  Cancela el scan actual.

```
>cancel()
```

  Activa los motores, argumento por omisión 'm1', las opciones son 'm1' y 'm2'.

```
>activate(motor)
```

  Devuelve el estado de los motores de los ejes de azimut y elevación.

```
>status()
```

  Lleva la antena a la posición de stow (45,88) e inserta los pines para bloquear la antena.

```
>stow()
```

Saca los pines de bloqueo de la antena y la lleva a posición de standby.

```
>unstow()
```

Aparca la antena en la posición (225,2) para poder acceder a la cabina de elevación.

```
>park()
```

Lleva la antena a máxima velocidad a las coordenadas (**az**,**el**), los valores por omisión son (45,88).

```
>gotoHor(az,el)
```

Introduce la orden **commLine** en el Field System

```
>fsCommand(commLine)
```

- **Configuración**

Selecciona si configurar la antena en foco primario o secundario. Los parámetros son 'primary' y 'secondary'

```
>fmode(focusMode)
```

Utilizar el DBBC o VLBA para configurar las atenuaciones de las señales de IF. 'True' por omisión.

```
>setDBBC(useDBBC)
```

Devuelve la temperatura del receptor para una configuración dada. Los parámetros posibles son,
frontend: 'HET2' 'HET8' 'HET22',
backend: 'idet', 'ffts', 'pbe'.

```
>trec(frontend,backend)
```

Carga los catálogos de fuentes.
catalogs: recibe una cadena de caracteres con la ruta del catalogo. Si se desea cargar varios catálogos introducirlos como una lista. Normalmente suelen estar ubicados en */home/raegemgr/raege/Catalogs*.
mode: 'keep' por omisión, mantiene los catálogos cargados anteriormente, 'new' no los tiene en cuenta, excepto para *system.cat* que viene cargado por omisión.

```
>sourcecats(catalogs,mode)
```

Escribe información en el archivo de log.
target: 0 por omisión, escribe en pantalla y en el log file. 1 solo en pantalla, 2 solo en el log.
color: False por omisión.

```
>logRaege(logline,target,color)
```

Carga los frontends indicados limpiando asignaciones anteriores de frontends y backends. Se pueden cargar varios frontends introduciendolos en forma de lista. Los frontends aceptados son 'het2', 'het8', 'het30'.

```
>frontends(names)
```

Elige los backends a conectar con el frontend seleccionado. Los backends de continuo solo se pueden conectar a un frontend. Usar con [['polarization','name',section] ,['polarization','name',section]].
polarization: 'rcp', 'lcp', 'h', 'v'.
name: 'idet', 'ffts', 'pbe'.
section: número indicando la sección usada para conectar la polarización especificada al backend. Actualmente en Yebes usamos: 1 y 2 para lcp y rcp en banda S, 3 y 4 para lcp y rcp en banda X y 5 y 6 para lcp y rcp en banda Ka.

```
>backends(self, backlist)
```

¡Sólo observaciones espectrales! Ajusta la configuración espectral del backend.
bwidth: ancho de banda [MHz].
nchan: número de canales.
firstChan: primer canal, 0 por omisión (pseudocontinuo).
lastChan: último canal, 0 por omisión (pseudocontinuo).
centralFreq: frecuencia central [MHz], 600 por omisión.

```
>setSetup(self, bwidth, nchan, firstChan, lastChan, centralFreq)
```

Selecciona el tiempo de integración en segundos, 1 por omisión.

```
>setIntegrationTime(self, integTime)
```

Define una posición de referencia para calibraciones y onoffs.
x:offset en longitud, 1000 por omisión.
y: offset en latitud, 0 por omisión.
reftime: tiempo de integración usado en posiciones de off, 2 por omisión.
on2off: número de ons por cada off, 1 por omisión.
unit: unidades de los offsets, 'arcsec' por omisión, otras posibilidades son 'arcmin', 'hms', 'dms', 'deg', 'rad'.
mode: 'rel' por omisión offsets relativos, 'abs' para offsets absolutos.
system: sistema de coordenadas usado para x e y, 'HO' por omisión, 'EQ' para ecuatoriales.
epoch: 2000 por omisión. solo válido para el sistema de coordenadas 'EQ'.
El offset usado en la referencia tiene que ser mucho mayor que el tamaño del haz, normalmente elegimos que sea unas tres veces y media superior. De esta forma el off no caerá dentro de la fuente. En el apéndice A se muestran los valores típicos usados en la referencia para las diferentes frecuencias del telescopio.

```
>reference(x,y,reftime,on2off,unit,mode,system,epoch)
```

Indica el proyecto en el que se van a guardar los datos del siguiente scan.
projectName: cadena de caracteres con el nombre del proyecto donde se guardaran los datos. El nombre de los proyectos de los usuarios suele tener el siguiente formato, *R-00.C-0001-2015*.
createDir: booleano indicando si se ha de crear un nuevo directorio para el proyecto en el caso en que éste no exista. False por omisión.

```
>project_id(projectName, createDir)
```

- **Observaciones**

Selecciona la fuente a observar.
name: cadena de caracteres con el nombre de la fuente del catálogo que esté cargado.
x e y: se pueden elegir las coordenadas indicando (coord,unit).
coord: en formato '12:34:56.789' o 12.345678.
unit: 'degs', 'hours', 'rads', 'hms', 'dms'.
system: sistema de coordenadas usado para x e y, 'HO' por omisión, 'EQ' para ecuatoriales.
epoch: 2000 por omisión. solo válido para el sistema de coordenadas 'EQ'.
velocity: velocidad radial de la fuente en km/s. None por omisión.
frame: sistema de referencia de la velocidad radial. 'LSR' por omisión.

```
>source(name,x,y,omx,pmy,system,epoch,velocity,frame)
```

Realiza un scan de calibración sobre la fuente y la posición de referencia.
mode: 'noisediode' por omisión.
timePerPhase: tiempo de integración en cada fase, 1 por omisión.

```
>calibrate(mode,timePerPhase)
```

Imprime por pantalla la temperatura del sistema de la última calibración para el frontend-backend especificado.
febeName: cadena de caracteres combinación del frontend y del backend separados por un guión. e.g: 'het2-idet'.
tastarscale: False por omisión indica tsys en la escala de Ta, True la indica en Ta*

```
>tsys(febeName,tastarscale)
```

Realiza un skydip.
az: azimut en el que realizar el skydip, por omisión el actual.
startel: en grados, la elevación donde empezar el skydip, 5 por omisión.
endel: en grados, la elevación donde terminar el skydip, por omisión 80.
points: numero de puntos donde la antena se para a integrar.
timePerPoint: en segundos, tiempo de integración por punto, 1 por omisión.

```
>skydip(az,startel,endel,points,timePerPoint)
```

Realiza un seguimiento de la fuente.
intTime: tiempo de integración en segundos, 30 por omisión.

```
>on(intTime)
```

Realiza un on en la fuente y un off en la referencia.
time: indica el tiempo dedicado al on, 30 segundos por omisión. El scan durará 2 x time (on+off) mas el tiempo de cambio de posición entre el on y el off.

```
>onoff(time)
```

Realiza un scan de foco, moviendo el subreflector entre startPos y endPos mientas sigue a una fuente.
startPos: en mm, -12 por omisión.
endPos: en mm, 12 por omisión.
axis: 'z' por omisión, otras posibilidades son 'x', 'y', 'tiltx' y 'tilty'. Las posiciones de los tilts se indican en segundos de arco.

```
>focus()
```

Retorna la corrección de foco requerida por el último scan de foco.
axis: 'x', 'y', 'z' por omisión, 'tiltx', 'tilty'.
febe: cadena de caracteres combinación del frontend y del backend separados por un guión. e.g: 'het2-idet'.

```
>getFocusFit(axis,febe)
```

Añade un offset al foco. Las correcciones siempre son relativas.
delta: numero de mm o segundos de arco que mover el foco.
axis: 'x', 'y', 'z' por omisión, 'tiltx', 'tilty'
correctPoint: True por omisión, corrige o no la la puntería debido a los cambios en el foco.

```
>fcorr(delta,axis,correctPoint)
```

Resetea a 0 los offsets añadidos por los usuarios.

```
>fcorr_reset()
```

Realiza un barrido sobre la fuente en azimut y elevación.
lenght: longitud del barrido.
unit: unidades en las que se ha indicado la longitud del barrido, 'arcsec'.
timePArm: en segundos, el tiempo de duracion de cada barrido.
mode:'otf' por omisión, 'ras' para raster.
pointsPArm: 10 por omisión.
cal: 'no' por omisión.
direction: dirección del barrido, 'x' por omisión,
zigzag: 0 por omisión, 1 realiza dos barridos.

```
>point(lenght,unit,timePArm,mode,pointsPArm,cal,direction,zigzag)
```

Retorna la corrección de puntería requerida por el último scan de puntería mediante un ajuste gaussiano con class.
vi y v2 : posición inicial de la ventana para pasar una linea de base.
v2: posición final de la ventana para pasar una linea de base.

```
>pointClass(v1,v2)
```

Añade un offset a la puntería.
azCol0: en segundos de arco, offset en azimut, 0 por omisión.
deltaEl0: en segundos de arco, offset en elevación, 0 por omisión.
mode: 'rel', 'abs', por omisión ignora las correcciones anteriores.

```
>pcorr(azCol0,deltaEl0,mode)
```

Resetea a 0 los offsets añadidos por los usuarios.

```
>pcorr_reset()
```

Realiza un mapa rectangular (on the fly).
xlen e ylen: Longitud del mapa en x e y en segundos de arco.
step: paso entre barridos .
refMode: 'abs' por omisión referido al centro del mapa, 'rel' referencia relativa al final del barrido.
direction: 'x' por omisión, '-x', 'y', '-y'.
zigzag: 0 por omisión, 1 para hacer los barridos bidireccionales.
unit: sistema de coordenadas usado para x e y, 'HO' por omisión, 'EQ' para ecuatoriales.
epoch: 2000 por omisión. solo válido para el sistema de coordenadas 'EQ'.

```
>otf(xlen,ylen,step,refMode,direction,zigzag,unit,system,epoch)
```

- **Funciones compuestas**

Realiza un scan de puntería y permite elegir si hacer correcciones o no.
pointLenght: Longitud del barrido en segundos de arco.
snrmax: si la señal a ruido está por debajo de este valor la puntería no se corrige.
maxcorr: corrección máxima de puntería permitida.
wmax: la mitad de la ventana usada para hacer un ajuste gaussiano.
correct: booleano, hacer o no correcciones de puntería.
doDrift: booleano, hacer o no doble barrido de puntería.
intTime: tiempo de integración en segundos.

```
>doPoint(pointLenght,snrmax,maxcorr,wmax,correct,doDrift,inTime)
```

Realiza un scan de foco, analiza los resultados y corrige el foco.
fmin: posición mínima del foco.
fmax: posición máxima del foco.
febe: cadena de caracteres combinación del frontend y del backend separados por un guión. e.g: 'het2-idet'.

```
>doFocus(fmin,fmax,febe)
```

- **Espera**

  Espera y bloquea el flujo de ordenes hasta que llegue el tiempo indicado.
  endEfTime: hora a la que hay que reanudar las acciones.
  logmode: 'silent' por omisión.
  verbose: manda información del estado de la antena mientras espera. 'no' por omisión, 'verbose'.
  verbperiod: periodo a usar para mostrar por pantalla el estado de la antena.

```
>waitabs(endEfTime,logmode,verbose,verbperiod)
```

  Espera y bloquea el flujo de ordenes el número indicado de segundos.
  ss: segundos que ha de esperar.
  logmode: 'silent' por omisión.
  verbose: manda información del estado de la antena mientras espera. 'no' por omisión, 'verbose'.
  verbperiod: periodo a usar para mostrar por pantalla el estado de la antena.

```
>waitrel(ss,logmode,verbose,verbperiod)
```

  Espera hasta que la antena esté disponible.
  timestep: tiempo de espera en segundo entre comprobaciones, 1 segundo por omisión.
  verbosity: False por omisión no se muestra ningún mensaje. True muestra el proceso por el que está esperando.

```
>waitUntilAvailable(timestep,verbosity)
```

- **Atmósfera**

  Muestra la opacidad, emisividad y la columna de vapor de agua para los frontends seleccionados.

```
>opacity()
```

  Muestra la información sobre el tiempo, temperatura, presión, humedad, velocidad del viento y dirección.

```
>wx()
```

- **Otros**

  Correr una macro de una observación.
  ruta: es un cadena de caracteres con la ruta de la macro.

```
>run(ruta)
```

  Si se desea parar una macro o cualquier otro proceso que esté corriendo, se pulsará *Ctrl+C* y la siguiente orden.

```
>cancel()
```

2.2. Macros

En este apartado se comentará la estructura que generalmente usamos en las macros para cualquier tipo de observación.

La primera parte de la macro se suele usar para configurar la antena y los receptores que vamos a utilizar, así como para definir algunas constantes que puedan ser útiles posteriormente. En el ejemplo que mostramos, se define el proyecto en el que se guardarán los datos, se cargan los catálogos de fuentes, se cargan los frontends, se define la frecuencia de observación, los backends, el tiempo de integración y la referencia para calibraciones. Seguidamente se definen las constantes que necesitemos usar posteriormente, generalmente se define el numero de ciclos del bucle principal, elevaciones mínimas y máximas de observación, longitud del barrido, ventana de corrección, febe, el rango de focos, modo de calibración... Finalmente se suelen reiniciar las correcciones de foco y puntería por si hubiera algunas cargadas previamente que no nos interesen y se añade una pequeña espera para que el subreflector pueda alcanzar la posición deseada.

```
project_id("R-00.C-0002-2014", True)
sourcecats(['/home/raegemgr/raege/Catalogs/user.cat'])
frontends("het8")
het8.line("VLBI8")
het8.backends([['lcp','idet',3],['rcp', 'idet', 4]])
idet.setIntegrationTime(1.0)
reference(-2400,0,8,1,'arcsec', 'abs','HO')

cyclesMax = 400
cycles = 0
elMin = 7.
elMax = 89.5
pointLength = 4000
wmax = 900
snrmax = 7
maxcorr = 150
febe = 'het8-idet'
calMode='noisediode'
calIntTime=15
fmin=-2-df1
fmax=-2+df1

fcorr_reset()
pcorr_reset()
waitrel(2)
```

A continuación se suelen añadir algunas funciones adicionales que queramos definir para su uso posterior en el bucle principal. Normalmente solemos crear una función para la calibración que busca la fuente y realiza otro intento de calibración en el caso de que la primera falle.

```
def doCal():
 global az, el, calMode, calIntTime, febe
 try:
  slts = gotoHor(az,el)
  waitrel(slts)
  waitrel(5)
  startTime, endTime = calibrate(calMode, calIntTime)
  waitabs(endTime)
  waitrel(5)
  ts = tsys(febe)
 except Exception, e:
  try:
   startTime, endTime = calibrate(calMode, calIntTime)
   waitabs(endTime)
   waitrel(5)
   ts = tsys(febe)
  except Exception, e:
   logRaege(str(e))
   waitrel(5)
```

Por último se añade el bucle principal, donde se indican las ordenes que se van a ir repitiendo en cada ciclo. Esta parte de las macros es muy diferente dependiendo de la observación que deseemos realizar, por ello, se explicará con detalle más adelante para cada tipo de observación. Ver apartados 4.1, 5.1 y 6.1.

3. Fuentes

Para un mejor y más rápido acceso a la información, todas las tablas y ecuaciones en esta sección se añaden al final del manual, en el apéndice B.

Si lo que queremos es realizar cualquier observación de puntería necesitaremos usar fuentes bastante intensas. Es necesario, por tanto, elegir adecuadamente las fuentes ya que no todas emiten de la misma forma en todas las frecuencias. En la tabla 2 se muestran las fuentes más intensas que podemos utilizar para realizar las observaciones de puntería. La mayoría de las fuentes que se muestran a continuación se han de cargar desde el catálogo '*user.cat*' (véase el apartado de configuración en 2.1).

En cuanto a las observaciones de ganancia, es necesario utilizar fuentes con un flujo y tamaño angular conocido. En la tabla 3, se muestra una lista de las fuentes más intensas con flujo conocido y los parámetros necesarios para realizar el cálculo del mismo. Estos parámetros se han obtenido del archivo '*flux.ctl*', que contiene muchas más fuentes y que encontraremos en la máquina '*fsraege*', en la siguiente ruta: */usr2/control/flux.ctl*. Para el cálculo del flujo de los planetas será necesario el uso del software ASTRO, dentro del paquete de GILDAS, ver 8.2.

Para el cálculo del flujo a partir de los parámetros utilizaremos la ecuación 7, en el apéndice, introduciendo los parámetros de la fuente y la frecuencia de observación deseada, siempre que ésta se encuentre dentro del rango de frecuencias válido de la tabla 3.

4. Observaciones de puntería

Este tipo de observaciones es recomendable realizarlas como mínimo 3 veces al año. Normalmente suele ser necesario cambiar los parámetros de puntería en verano y en invierno ya que los principales errores que existan suelen deberse a cambios de temperatura estacionales. Las correcciones que se aplican a la antena son dependientes de 8 parámetros de un modelo que ajusta los errores en función de varias funciones trigonométricas de la elevación y el azimut. El objetivo, por tanto, será realizar observaciones que nos permitan obtener datos para el ajuste de los parámetros de este modelo.

4.1. Observación

La determinación de los errores de puntería en acimut y elevación requieren una cobertura lo más completa posible del cielo. Para ello serán necesarias unas 24 horas de observación de fuentes con diferentes coordenadas. El número de fuentes disponibles para la antena del RAEGE de 13,2 m es más limitado que para el radiotelescopio de 40 m, debido a su menor área colectora. Para disponer de una buena relación señal a ruido sólo se pueden utilizar las fuentes más intensas del cielo.

Las fuentes que se suelen utilizar, las podemos obtener de la tabla 2 en el apéndice B. Resultará conveniente realizar las observaciones en la frecuencia más alta posible ya que así tendremos una precisión mayor en el cálculo de los errores de puntería (suponiendo que las bocinas del receptor tribanda están centradas y por tanto tienen los mismos errores). Una vez seleccionadas las fuentes que observaremos, podremos comprobar que cobertura tenemos, por ejemplo, con el script de Python *coberturaSMAzores.py* realizado por Antonio Díaz Pulido. A modo de ejemplo, se muestra en la figura 1, la cobertura típica que obtenemos en una observación en banda X con el RAEGE de Yebes.

El proceso de observación consta básicamente de una calibración al principio de cada ciclo (no es importante calibrar para conocer los errores en puntería), y una puntería en cada una de las fuentes a observar, de esta manera se consiguen observar todas las fuentes a lo largo de su trayectoria en el cielo de manera continua. No se deben aplicar correcciones de puntería durante el proceso, porque precisamente estamos buscando errores en puntería. En el caso de tener errores de puntería tan grandes que las fuentes no son visibles en los barridos, será necesario aplicar unas correcciones de puntería fijas, teniendo en cuenta que una vez obtenidos los errores tendremos que deshacer el cambio realizado. A continuación se muestra el bucle principal para una observación típica de puntería a 8 GHz.

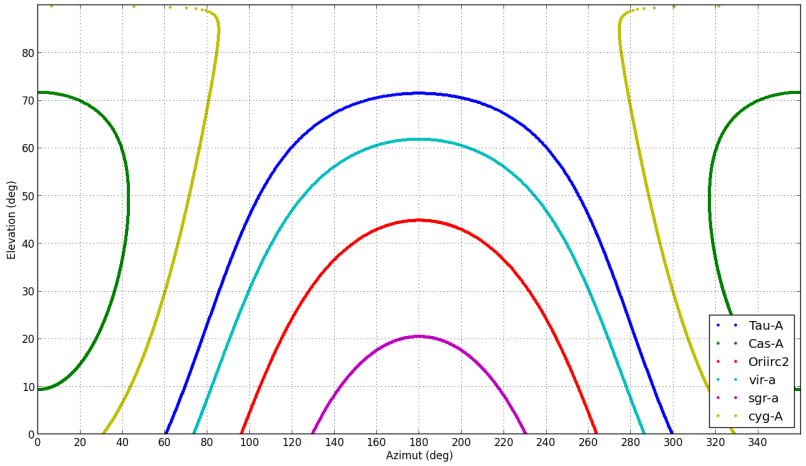

Figura 1: *Cobertura típica en una observación de banda X con el RAEGE de Yebes.*

```
while cycles < cyclesMax:
    calibra=True
    lastsource="None"
    logL="\n***Loop Cycle/Loop Max Cycles: %d/%d***" %(cycles, cyclesMax)
    logRaege(logL)
    sourceList=["Tau-A","OriIRC2","VIR-A","Sgr-A","Cyg-A","Cas-A"]
    for sourceName in sourceList:
        (az,el)=source(sourceName)
        if el > elMin and el < elMax:
            if calibra==True:
                doCal()
                calibra=False
            lastsource=sourceName
            try:
                sT,eT=point(pointLength,'arcsec',60,'otf',10,'no','x',1)
                waitabs(eT)
                waitrel(10)
            except Exception, e:
                strEx = str(e)
                logRaege(strEx)
                waitrel(5)

    if lastsource=="None":
        logL = "!!!!All sources below elevation limit:%f!!!"%elMin
        logRaege(logL)
        waitrel(120)

    cycles = cycles +1
```

4.2. Análisis de las observaciones

Una vez obtenidas punterías en todas las regiones del cielo durante 24 h se analizan los datos. Se trata de generar los archivos tipo *point* y *fit* con CLASS a partir del archivo con los datos de puntería, tal y como se describe en la sección 8.1. Estos archivos contendrán todos los datos necesarios para el análisis de las punterías, coordenadas, errores de puntería, tipo de barrido...

Existe un pequeño error al crear los datos, y es que el código que identifica si un barrido es de azimut o elevación no tiene el mismo valor en los archivos que genera CLASS y en los que lee el programa para el cálculo de los parámetros del modelo. Actualmente CLASS identifica con un 0 los barridos de azimut y con un -1 los de elevación, mientras que el *OAN 40m Pointing Model* toma el 1 como un barrido de azimut y el 0 como uno de elevación. Hay varias formas de solucionar este problema, empezando por cambiar el código del programa, pero nosotros hemos optado por modificar el archivo que retorna CLASS para adaptarlo al *OAN 40m Pointing Model*.

El siguiente paso será combinar ambos archivos para que tengan un formato adecuado que el programa '*OAN 40m Pointing Model*' permita leer. Hemos escrito un par de scripts en Python que permiten analizar los datos con CLASS y realizar las operaciones anteriores obteniendo así el archivo que le hemos de pasar al '*OAN 40m Pointing Model*'. El primer script que genera los archivos *point* y *fit* se llama '*PointFitFile.py*'. Una vez obtenidos estos archivos podremos ejecutar el '*createPointModelFile.py*', con el que combinaremos los archivos y cambiaremos el código de los barridos que ya hemos mencionado para obtener un único archivo que pueda ser reconocido por '*OAN 40m Pointing Model*'.El apéndice C contiene más información sobre estos programas de análisis.

Con el archivo de datos debidamente combinado, ya estamos listos para ejecutar el '*OAN 40m Pointing Model*'. Cuando leemos el archivo con el programa, lo primero que podemos ver es una gráfica de azimut frente a elevación donde podemos comprobar si tenemos una buena cobertura del cielo. A continuación, ejecutaremos el cálculo del modelo (pestaña *execute*) y podremos comprobar como ha salido. En la pestaña *chart* podremos seleccionar diferentes gráficas (ver la figura 2) representando los errores y las coordenadas en azimut y elevación tanto de los datos (puntos negros), como de las predicciones (puntos rojos).

El siguiente paso será eliminar los puntos que tengan valores inconsistentes ya que se puede haber colado alguna mala puntería. para ello podemos desmarcar los puntos que queramos en la pestaña *Edit/Observations*. En la figura 3 se muestra un fragmento de la lista con todos los datos junto con la discrepancia respecto al modelo ajustado. Este último valor también puede resultar de utilidad para descartar algunos puntos. Una vez eliminadas las punterías que no sean buenas, podemos volver a ejecutar el cálculo de los parámetros y podemos asumir que éstas son las correcciones que necesita la antena. Hay que comentar que los parámetros que se muestran en el programa son del p1 al p5 y del p7 al p9 ya que el p6 no es necesario. Si se desea es posible exportar los resultados a un archivo pdf con la opción *File/Export Results*. Hay que tener en cuanta, además, que los parámetros que obtenemos del programa son acumulativos por lo que deberemos sumárselos a los que ya estén cargados en la antena.

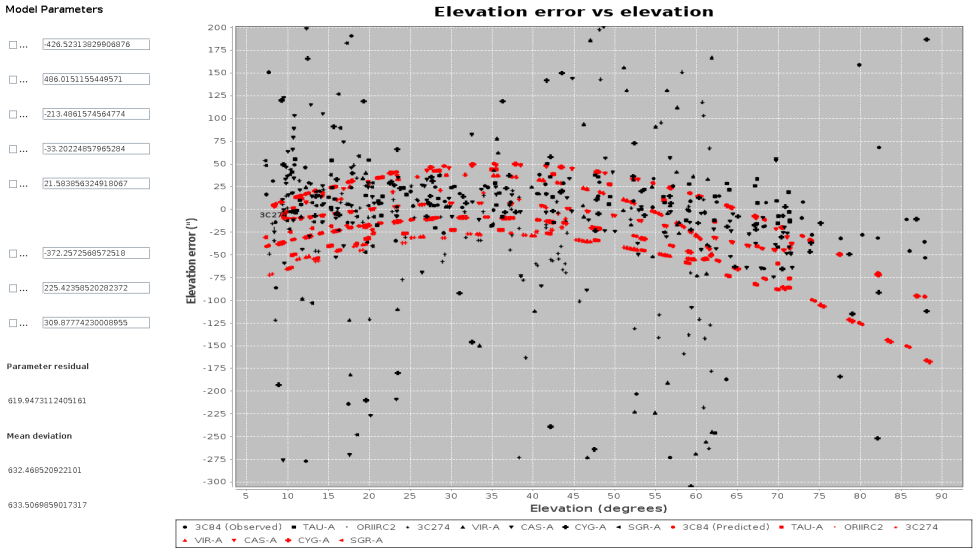

Figura 2: *OAN 40m Pointing Model. Error en elevación frente a elevación*

Observations

Fitted ?	File name	Source	Azimuth (deg)	Elevation (deg)	Azimuth erro...	Elevation err...	Fit discrepan...
☑		3C84	305,491	23,385	-689		642,013
☑		3C84	305,491	23,385	28,2		16,275
☑		3C84	305,746	23,179	1.280		1.166,905
☑		3C84	305,746	23,179	-7,34		16,521
☑		3C84	305,841	22,925		685	644,493
☑		3C84	305,841	22,925		923	882,493
☑		3C84	305,993	22,826		-795	835,351
☑		3C84	305,993	22,826		604	563,649
☑		TAU-A	269,657	35,584	-295		241,733
☑		TAU-A	269,657	35,584	-1,33		2,902
☑		TAU-A	269,991	35,33	-751		614,407
☑		TAU-A	269,991	35,33	45,5		35,405
☑		TAU-A	270,141	35,028		-1.330	1.374,136
☑		TAU-A	270,141	35,028		-17,7	61,836
☑		TAU-A	270,354	34,884		357	312,802
☑		TAU-A	270,354	34,884		-18,4	62,598
☑		ORIIRC2	247,383	16,637	581		550,101
☑		ORIIRC2	247,383	16,637	-81,6		84,761
☑		ORIIRC2	247,731	16,403	832		791,065
☑		ORIIRC2	247,731	16,403	-48,6		53,693
☑		ORIIRC2	247,916	16,118		-769	779,673

Figura 3: *OAN 40 m Pointing Model. Edit/Observations*

5. Observaciones de foco

Al igual que ocurre con la puntería la posición óptima del foco de la antena puede cambiar con el tiempo. Las variaciones del foco son algo más inusuales que las de la puntería por lo que nos bastará con repetirlas una vez por año o si notamos que la eficiencia de la antena ha bajado considerablemente. Usualmente solo se comprueba la posición del foco en Z y aunque no suelen ocurrir variaciones en X e Y no está de más comprobarlo de vez en cuando. Con la antena de 40 m es usual antes de los onoffs o punterías ejecutar la función '*doFocus()*'. Esta función mueve el subreflector entre dos posiciones determinadas y determina en que punto la intensidad de la señal recibida es mayor para corregir la posición del foco. Con la antena del RAEGE tenemos un problema con esta función ya que el "spill-over" aumenta considerablemente mientras el subreflector se acerca a la parábola no pudiendo determinar un máximo real en la intensidad debido al foco. Por ello, con la antena del RAEGE no podremos usar esta función. En su lugar, lo que vamos a hacer va a ser realizar punterías sobre una misma fuente con diferentes posiciones del foco a lo largo del eje axial (Z) y a diferentes elevaciones.

5.1. Observación

A continuación se muestra el bucle principal de una macro típica de foco. Lo ideal es elegir una fuente que sea muy intensa y cubra el mayor rango de elevaciones posible. En esta macro hemos introducido una lista de fuentes por orden de preferencia, la observación se realizará sobre la primera fuente siempre que esté disponible, una vez la fuente se oculte la antena cambiará a la siguiente fuente.

La macro realiza ciclos de calibración, puntería para corregir los errores y un ciclo de punterías en varias posiciones del subreflector. En este caso son, 5 punterías entre -20mm y +20mm con pasos de 10mm.

```
def focusLoop(foco):
    global pointLength, snrmax, maxcorr, wmax
    fcorr_reset()
    fcorr(-20,foco)
    waitrel(10)
    doPoint(pointLength, snrmax, maxcorr, wmax, False, True, 1)
    for i in range(0,5):
        fcorr(+10,foco)
        waitrel(15)
        doPoint(pointLength, snrmax, maxcorr, wmax, False, True, 1)
        fcorr_reset()
        waitrel(10)
#-----------------Main loop-----------------
sourceList=["Tau-A","Cas-A","Cyg-A","J1230+123","J0319+415"]
foco='z'
while cycles < cyclesMax:
    logL="\n***Loop Cycle/Loop Max Cycles: %d/%d***" %(cycles, cyclesMax)
    logRaege(logL)
    entrado=False
    for sourceName in sourceList:
        (az,el)=source(sourceName)
```

```
    if el > elMin and el<elMax and not entrado:
    entrado=True
    try:
        doCal()
        doPoint(pointLength, snrmax, maxcorr, wmax, True, False,1)
        focusLoop('z')
    except Exception, e:
        strEx = str(e)
        logRaege(strEx)
        waitrel(2)
if lastsource=="None":
    logL = "!!!!All sources below elevation limit:%f!!!"%elMin
    logRaege(logL)
    waitrel(120)
cycles = cycles +1
```

5.2. Análisis de las observaciones

Una vez realizadas las observaciones tendremos que obtener con CLASS el archivo *point* con los datos de temperatura de cada puntería (sección 8.1). Para obtener la información sobre la posición que tenía el subreflector en cada puntería podemos usar el script '*getzfocusRaege.py*' que se conecta a la base de datos y retorna un archivo '.txt' con información sobre el numero de scan, coordenadas, posición del foco... A continuación se utiliza otro script denominado '*focusFile.py*' que recorre todos los datos de los dos archivos que hemos generado y los combina.

Con esta observación se obtendrán varias curvas de ganancia, una para cada posición del subreflector. De esta manera podemos representar los datos de temperatura de antena frente a elevación o incluso, frente a la posición del subreflector. La posición óptima es la que produce mayor ganancia (ver figura 4).

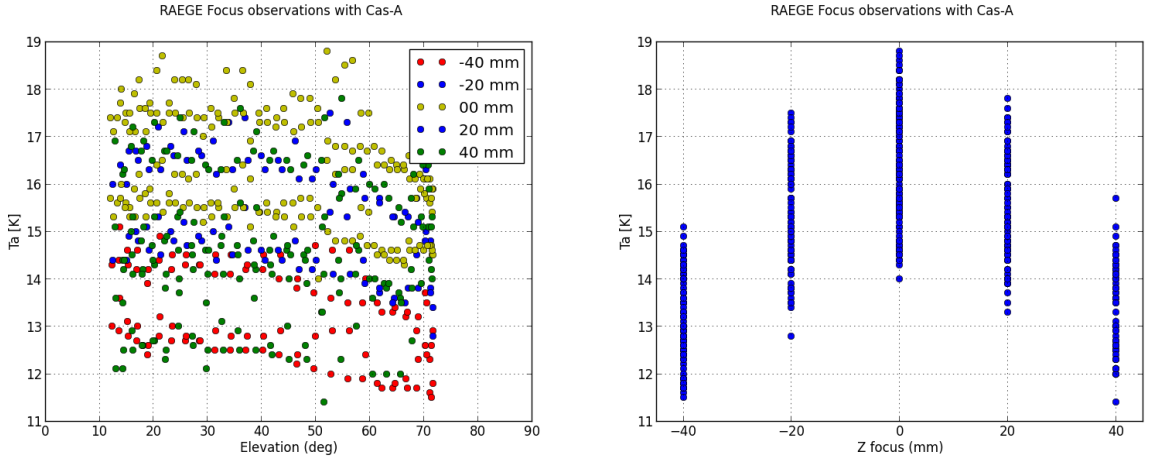

Figura 4: *Izda: Temperatura de antena frente a elevación con diferentes posiciones del subreflector. Derecha: Temperatura vs posición del subreflector.*

6. Observaciones de ganancia

El objetivo de estas observaciones es conocer la eficiencia delantera de la antena y su dependencia con la elevación. En este tipo de observaciones es necesario conocer perfectamente el flujo de la fuentes que se está observando. Existen algunas fuentes de referencia que se suelen usar para realizar este tipo de observaciones. En la tabla 3 del apéndice B se muestran las fuentes más usadas para cada frecuencia.

6.1. Observación

Este va a ser el tipo de observación más sencillo que vamos a planear. En las macros de ganancia lo único que necesitamos es tener punterías bien calibradas de una misma fuente en un rango muy amplio de observaciones. Por ello, primero resultará bastante útil utilizar algún programa como el KStars o el script '*eleVStime.py*' programado por Antonio Díaz Pulido para consultar la posición que alcanzan varias fuentes en el cielo. A continuación se muestra en la figura 5 una captura de este script donde se muestra a lo largo de 24 h la elevación de CAS-A, TAU-A y CYG-A en función del tiempo. Generalmente utilizaremos un par o tres fuentes en la macro para que cuando se oculte alguna de ellas, siempre tengamos otra disponible y que la antena no esté parada. De esta forma vamos a poder obtener un par de curvas de elevación en una misma noche.

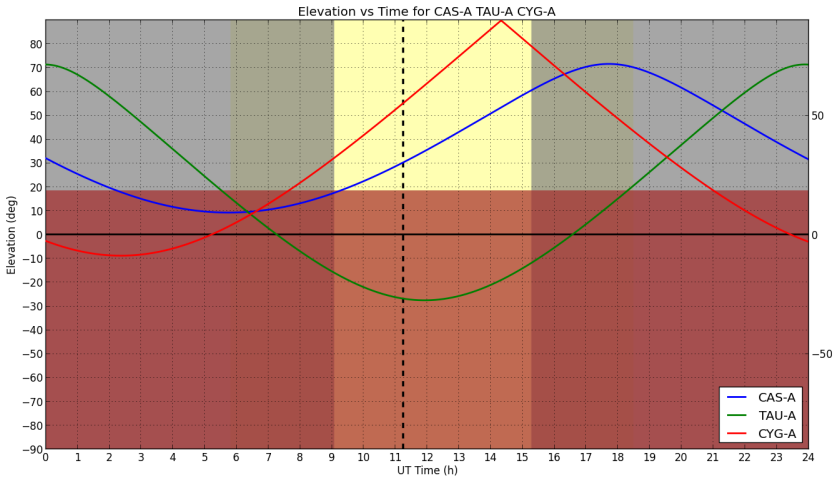

Figura 5: *Muestra del script 'eleVStime.py' para la planificación de observaciones,*

En cuanto al bucle principal de las observaciones, la estructura es similar a la de las observaciones de foco en las que seguimos una misma fuente hasta que se oculta, pero en este caso realizaremos una calibración y un par de punterías corrigiendo los errores. Con el 40 m solemos realizar también correcciones de foco aunque ya hemos explicado en el apartado 5, que con el RAEGE, el spill-over nos impide utilizar la función focus().

```
sourceList=["TAU-A","Cas-A","CYG-A"]
while cycles < cyclesMax:
   logL="\n***Loop Cycle/Loop Max Cycles: %d/%d***" %(cycles, cyclesMax)
   logRaege(logL)
   entrado=False
   for sourceName in sourceList:
      (az,el)=source(sourceName)
      if el > elMin and el<elMax and not entrado:
         entrado=True
         try:
            doCal()
            doPoint(pointLength, snrmax, maxcorr, wmax, True, False,1)
            doPoint(pointLength, snrmax, maxcorr, wmax, True, True,1)
         except Exception, e:
            strEx = str(e)
            logRaege(strEx)
            waitrel(2)
   if lastsource=="None":
      logL = "!!!!All sources below elevation limit:%f!!!"%elMin
      logRaege(logL)
      waitrel(120)
   cycles = cycles +1
```

6.2. Análisis de las observaciones

Con los datos ya generados en el archivo '.40m' podemos generar listas de datos con la temperatura de antena de las punterías y la elevación con el script '*PointFitFile.py*'. Conviene señalar que este script también permite obtener los datos separados por polarizaciones y por tipo de barrido para comprobar defectos en las ganancias debidos a errores de puntería o a diferencias entre las polarizaciones. Podríamos ver, por ejemplo, que si tenemos errores de puntería en elevación los barridos de azimut tendrían una intensidad menor que los de elevación.

A continuación debemos convertir la temperatura de la antena que obtenemos con CLASS a eficiencia de apertura, para ello necesitaremos la ecuación 1.

$$\eta_a = \frac{2K_B C_s T'_a}{AS_f} \tag{1}$$

Donde, K_B es la constante de Boltzmann, C_s es la función de convolución del haz y la fuente, T'_a es la temperatura de antena estrella corregida por los efectos de la membrana y la eficiencia delantera, que en el caso del RAEGE podemos considerar prácticamente del 100 %. A es el área colectora de la antena y S_f el flujo de la fuente observada. Para una antena de 13.2 metros podemos simplificar la ecuación 1 obteniendo:

$$\eta_a = 20{,}168\frac{C_s T'_a}{S_f} \tag{2}$$

El flujo de la fuente lo podemos encontrar en la tabla 3 del apéndice B, en el archivo '*/usr2/control/flux.ctl*' o en ASTRO (ver sección 8.2) en el caso de que sea un planeta. Para calcular el factor de convolución entre el haz y la fuente debemos conocer el tamaño de la fuente (θ_s) también en ASTRO para planetas y en B o '*flux.ctl*' para otras fuentes. El tamaño del haz (θ_b), lo encontraremos en el apéndice A o se puede calcular conociendo el diámetro de la antena y la longitud de onda de observación con la ecuación 11 del apéndice E

$$C_s = \begin{cases} 1 + x^2 & \text{Fuente gaussiana} \\ \frac{x^2}{1-e^{-x^2}} & \text{Fuente disco} \end{cases} \tag{3}$$

donde,

$$x = \frac{\theta_s(''')}{1{,}2\theta_b(''')} \tag{4}$$

Una vez convertida la temperatura de antena a eficiencia de apertura ya estamos en condiciones de preparar gráficas de eficiencia frente a elevación. En la figura 6 se muestra un ejemplo de una de estas curvas en banda X.

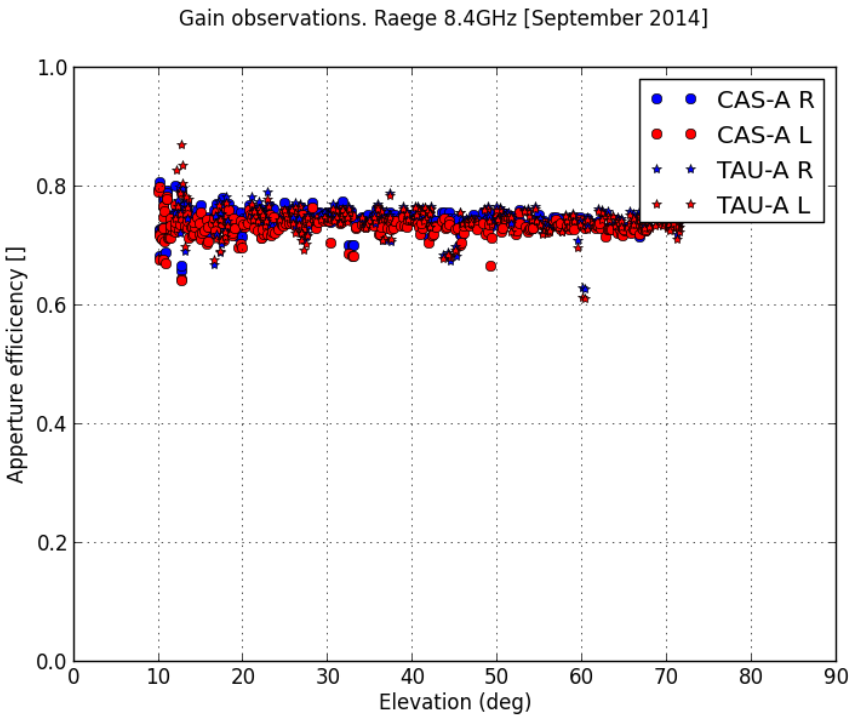

Figura 6: *Muestra de curva de ganancia en banda X.*

7. Skydips

Para conocer el estado de la atmósfera es conveniente realizar skydips. Con ellos podremos estimar la opacidad atmosférica (τ_0) así como algunos de los parámetros de la antena como la eficiencia delantera (η_f) o la temperatura del receptor (T_{rx}). En un skydip la antena toma datos apuntando al cielo en continuo entre los 5 y los 88 grados. Los datos obtenidos se guardan en un archivo '.dat' con 4 columnas. La primera son datos de las variaciones en azimut de la antena que suelen ser muy pequeñas. La segunda columna son los datos de la elevación de la antena menos 5 grados. Las columnas 3 y 4 representan respectivamente las temperaturas de las polarizaciones derecha e izquierda.

La temperatura de antena del cielo durante un skydip será la temperatura del receptor más la del suelo (debido al spill-over) y la del cielo que tendrá una dependencia con la elevación y la opacidad. El comportamiento, por tanto, se puede definir con la ecuación 5.

$$T^a_{sky} = \eta_f T_{atm}(1 - e^{-\tau_0/sin(el)}) + T_{rx} + T_g(1 - \eta f) \tag{5}$$

Donde T_g es la temperatura del suelo, equivalente a la temperatura ambiente y T_{atm} representa la temperatura de la atmósfera que suele ser la temperatura ambiente menos 40º.

7.1. Análisis de las observaciones

Para obtener datos útiles de los skydips será necesario realizar un ajuste por mínimos cuadrados de la ecuación 5. Si conocemos el valor de la temperatura del receptor, podremos realizar un ajuste a dos parámetros para obtener la eficiencia delantera y la opacidad. Incluso cabe la posibilidad de que conozcamos la eficiencia delantera con lo que el skydip nos servirá para obtener con bastante precisión los valores de la opacidad atmosférica. También conviene saber que aunque las dos polarizaciones presenten valores diferentes de la temperatura del receptor o incluso de la eficiencia delantera, y los skydips estén escalados de forma diferente, la forma de ambas curvas ha de ser la misma ya que la opacidad atmosférica no puede variar según la polarización.

```
import scipy.optimize as scp
import numpy as np
x=np.loadtxt('skydip_53008_HET45-PBE.dat',unpack=True)
par=np.array([0.01])
def skydip(elev,o):
 e=0.9
 Tg=20
 Trx=60
 Tatm=Tg-40
 return (1-e)*(Tg)+Trx+e*(Tatm)*(1-np.exp(-o/np.sin(elev*np.pi/180.)))
parrcp=scp.curve_fit(skydip,x[1],x[2]-min(x[2]),par)
```

Hay varias formas de realizar un ajuste por mínimos cuadrados pero una forma bastante fácil es mediante los módulos de Python 'scipy.optimize' y 'numpy'. Arriba se muestra un ejemplo de un ajuste de un skydip a un parámetro (opacidad). Primero cargamos los datos en listas y definimos un valor inicial para el parámetro que queremos ajustar a modo de lista.

A continuación se ha de definir la función que queremos ajustar dejando como variables el parámetro a ajustar (la opacidad en este caso) y el valor de la abscisa (elevación). Finalmente el ajuste de los parámetros se genera al ejecutar la función 'curve_fit' del paquete 'scipy.optimize'. Como variables le tendremos que pasar el nombre de la función que hemos definido, un lista con los datos en abscisas, en ordenadas y el valor inicial de los parámetros a ajustar.

Por último, nos queda decir que tenemos un script de Python preparado para realizar ajustes de skydips y pintar gráficos con los datos y la curva teórica. Se puede consultar información sobre este script en el apéndice C.

8. Reducción de datos con GILDAS

GILDAS es un paquete de software orientado a la reducción de datos de observaciones radioastronómicas. GILDAS contiene el software GREG, SIC, CLASS, ASTRO... Para todas las observaciones anteriores solo necesitaremos conocer el funcionamiento de CLASS y ASTRO, si queremos hacer mapas también será necesario el uso de GREG.

8.1. CLASS

CLASS es el software que utilizamos en todas nuestras observaciones para la reducción y el análisis de los datos. CLASS lee y permite procesar los archivos con formato '.40m' que se generan automáticamente a partir de los scans '.fits'. A continuación mostraremos una lista con las funciones más usadas para pasar posteriormente a explicar el proceso de reducción de los datos.

8.1.1. Funciones CLASS

Aquí mostramos una lista de las funciones más útiles en CLASS. Los argumentos se especifican separados por espacios, ejemplo: CAS90>file in datos.40m.

Iniciar el programa CLASS.

```
>class
```

Carga un archivo de entrada con formato '.40m'

```
>file in entrada.40m
```

Carga un archivo de salida para CLASS.

```
>file out salida.40m multiple
```

Para configurar un valor de uno de los parámetros de CLASS utilizaremos el comando >set seguido del parámetro a modificar. Si introducimos la orden y el parámetro sin ningún valor reiniciaremos el valor . Los mas utilizados son los siguientes:

Especifica el tipo de observación. Los argumentos son 'cont' para continuo 'line' para espectral.

```
>set type
```

Especifica la fuente en la que estamos interesados para filtrar el resto.

```
>set source
```

Especifica la frecuencia central de observación en la que estamos interesados para filtrar el resto.

```
>set line
```

Especifica la configuración del telescopio en la que estamos interesados para filtrar el resto.

```
>set telescope
```

Especifica el scan o rango de scanes si se especifica valor inicial y final.

```
>set scan
```

Especifica el modo del plot, 'n' para normal y 'h' para histograma.

```
>set plot
```

Especifica el formato de la cabecera de los plots, 'full' para mostrar toda la información y 'brief' para no mostrarla.

```
>set format
```

Selecciona los límites inferior y superior para la ventana de extracción. Es necesario seleccionar este parámetro para pasar una linea de base a las observaciones y realizar un ajuste gaussiano. Si tecleamos este comando con un plot abierto y sin pasarle ningún parámetro podremos introducir los límites de manera gráfica.

```
>set window
```

Encuentra todos los scans con los parámetros que hemos definido.

```
>find
```

Lista todos los scans que se han encontrado con el último find.

```
>list
```

Lista las fuentes, frecuencias y configuraciones de telescopio disponibles. Además podemos añadir 'source', 'line', o 'telescope' para que solo aparezca la lista de una característica en concreto.

```
>list /toc
```

Selecciona un scan concreto de la lista. Las opciones son 'f' para el primero, 'n' para el siguiente, 'p' para el anterior y 'l' para el último.

```
>get
```

Muestra el valor de la variable elegida. Si no se define una variable se mostrará la lista de las variables disponibles.

```
>examine
```

Plotea el scan seleccionado.

```
>plot
```

Sustrae la linea de base de la observación obviando las zonas previamente seleccionadas mediante 'set window'. Se le ha de pasar como parámetro el grado del polinomio al que queremos ajustar la linea de base, normalmente grado 1 o 2.

```
>base
```

Realiza un ajuste gaussiano del scan actual y muestra el resultado por pantalla.

```
>fit \min
```

Plotea el resultado del ajuste gaussiano.

```
>fit\vis
```

Escribe los resultados del último ajuste en el archivo de salida actual.

```
>write
```

Escribe los datos de los ajustes de todos los scans encontrados con el último find en el archivo 'ajuste.txt'. Los datos que se guardarán son (1) número de la linea, (2) número de observación CLASS, (3,4) offsets en segundos de arco, (5,6) área de la linea y incertidumbre, (7,8) posición y incertidumbre, (9,10) anchura de la linea y incertidumbre, (11) intensidad, (12) rms de la linea de base, (13) rms de la linea.

```
>print fit /output ajuste.txt
```

Escribe los datos de los ajustes de todos los scans encontrados con el último find en el archivo 'ajuste.txt'. Los datos que se guardarán son (1) número de observación de CLASS, (2) número de scan, (3) dirección del barrido, (4,5) azimuth y elevación en grados, (6) tiempo en horas, (7,8) posición y error en la dirección del barrido, (9,10) número de la antena y codigo de la estación, (11,12) anchura y error, (13,14) intensidad y rms, (15) nombre de la fuente.

```
>print pointing /output ajuste.txt
```

Realiza un bucle para todas las observaciones encontradas con find, empezando por la número 1. 'Pause' detiene el bucle que se reanudará pulsando c. Next indica el final del bucle.

```
>for i 1 to found
>get n
>plot
>pause
>next
```

Muestra información sobre la función deseada.

```
>help
```

8.1.2. Reducción de observaciones en continuo

En la mayoría de los casos, siempre que queramos realizar la reducción de los datos, vamos a realizar siempre las mismas acciones. Aquí se detalla el orden a seguir para la obtención de un ajuste gaussiano de unos datos que bien nos podría servir para realizar curvas de ganancia o reducir observaciones de puntería.

En primer lugar tendremos que iniciar el programa en la carpeta donde tengamos el archivo '.40m' y lo cargaremos. Con el CLASS ya abierto podemos ver que el prompt nos muestra LAS90>, esto indica que estamos en modo espectral (line), por tanto cambiaremos a modo continuo con '>set type c'. Seguidamente podemos hacer un '>find' para buscar las observaciones que tenemos en el archivo y un '>list' para que nos las muestre. En la lista, podemos observar el número de observación de CLASS en la primera columna y el scan en la penúltima. A cada scan de puntería en continuo le corresponderán 4 o 8 observaciones,2 barridos por 2 polarizaciones o 4 barridos por 2 polarizaciones en el caso de que sean punterías dobles. Si lo deseamos,

podemos filtrar por fuentes, frecuencia, configuración (telescope) o scan. Podemos seleccionar el primer scan por ejemplo con '>get f' o con '>get 1' donde el número indicado es el de la observación, no el del scan. Una vez seleccionado podremos mostrar la puntería por pantalla con '>plot'. Como vemos en la figura 7, en el plot se muestra información sobre la observación seleccionada, podemos decidir ocultarla o mostrarla mediante el comando '>set format' o cambiar al modo histograma con '>set plot h'.

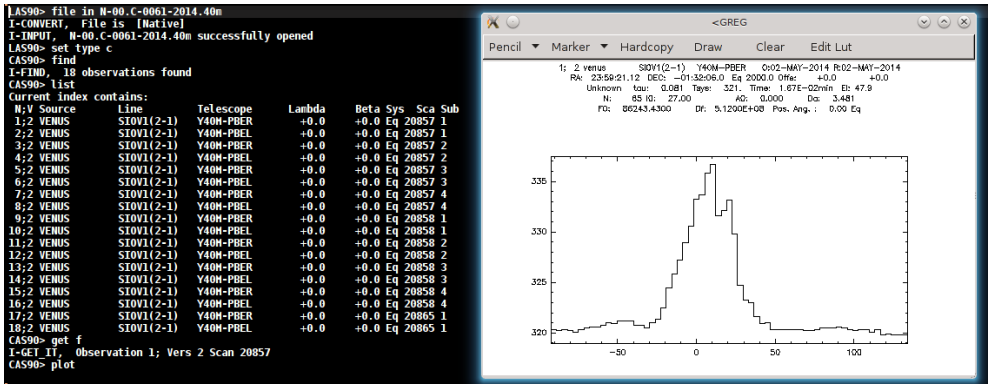

Figura 7: *Primeros pasos en CLASS.*

Una vez tenemos seleccionado el scan podemos pasarle una linea de base. Para ello primero seleccionaremos una ventana de extracción con '>set window' y a continuación pasaremos la linea de base '>base 1'. Si volvemos a plotear los datos veremos como la posición de la base se ha desplazado hasta 0 K, figura 8. Podemos guardar los datos

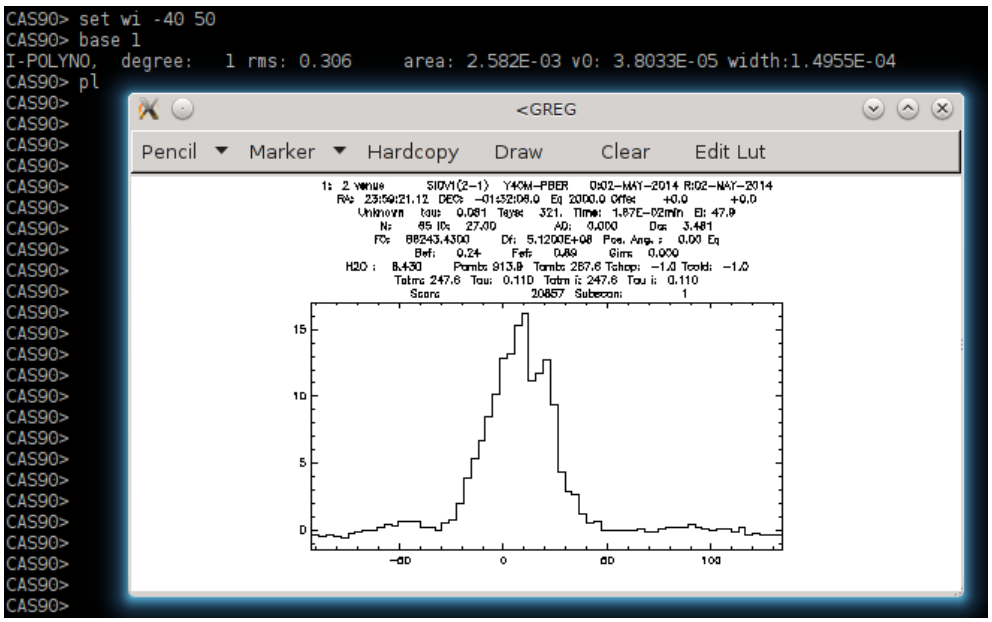

Figura 8: *Pasar una linea de base.*

El siguiente paso será realizar un ajuste de la puntería a una función gaussiana. Para ello realizaremos el ajuste automáticamente con el comando '>fit\min' y lo visualizaremos con '>fit\vis' como podemos ver en la figura 9.

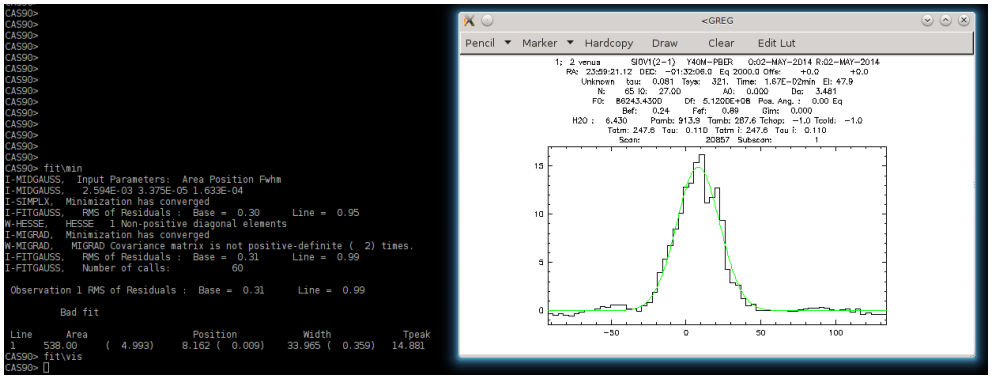

Figura 9: *Ajustes con CLASS.*

Por último si queremos guardar los datos del ajuste y de la linea de base necesitaremos definir un archivo de salida con '>file out' y introducir el comando '>write' después de realizar cada ajuste. Los datos se guardarán en un archivo con formato '.40m'. Finalmente, para almacenar los datos en un archivo '.txt' con la información separada por columnas utilizaremos el comando '>print fit /output salida.txt' o 'print pointing /output salida.txt' según la información que necesitemos. Hay que tener en cuenta que para las observaciones de puntería vamos a necesitar los dos archivos, véase 4. Para realizar este proceso de forma repetitiva para todos los datos va a ser necesario introducir un bucle for de CLASS. Falta recordar que el script '*PointFitFile.py*' está preparado para repetir este proceso de forma automática, por tanto, puede ser útil repasar su funcionamiento para conocer mejor los pasos a seguir.

8.2. ASTRO

Astro es un software desarrollado para ayudar a astrónomos a preparar las observaciones, principalmente para el cálculo de efemérides. Aunque el programa permite muchas más opciones, nosotros solemos utilizarlo para obtener datos de los planetas y información meteorológica. Para obtener el flujo, tamaño o coordenadas de un planeta en un momento determinado necesitaremos especificar la situación del observatorio la frecuencia deseada y la hora de observación. Los comandos que le introduciremos a ASTRO serán:

```
>observatory 36.981195 -25.132474 0.215
>time 10:00:00 26-05-2014
>line "7mm" 45 usb
>planet
```

Primero se introducen las coordenadas del observatorio (latitud, longitud y altitud) a continuación introduciremos la hora y fecha de observación con el comando time, y por fin se introducirá la frecuencia de observación con line. Si introducimos el comando '>planet' se nos

mostrará información de todos los planetas como coordenadas, distancia angular al sol, tamaños de los semiejes mayor y menor, flujo... En la figura 10 se puede ver la información mostrada de los planetas.

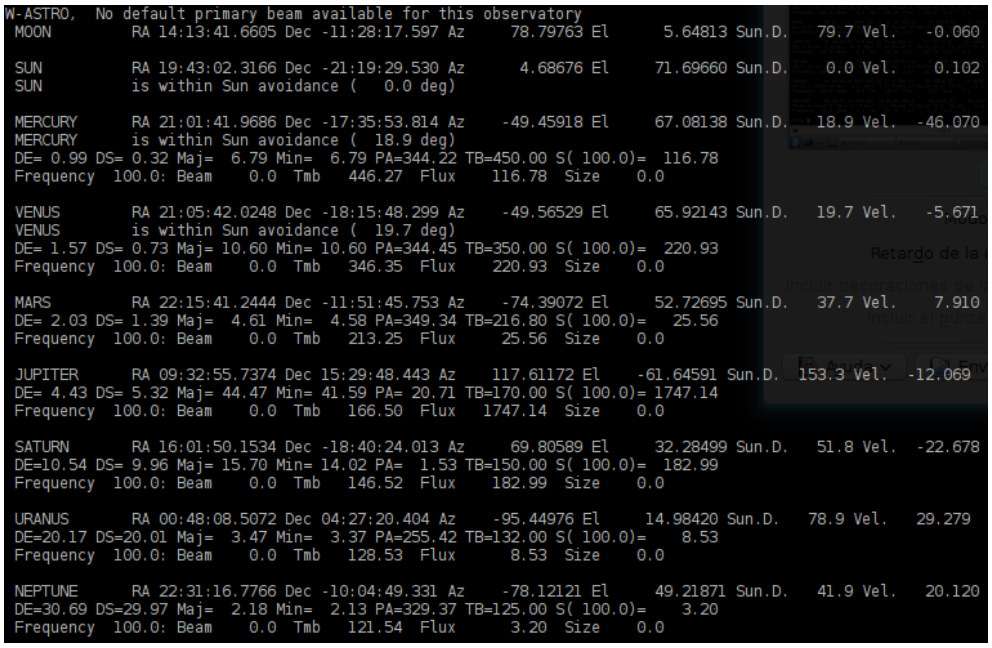

Figura 10: *Información de los planetas con ASTRO.*

Además también es posible adquirir información sobre la opacidad atmosférica mediante el paquete de funciones ATM que también nos proporciona ASTRO. Para ello hemos de introducir el observatorio de la misma manera que hemos hecho antes y definir, con el comando 'let', los valores de temperatura, columna de vapor de agua, masa de aire, frecuencia señal e imagen... Además tendremos que introducir la presión a nivel del mar, para ello será necesario conocer el factor de escala (h_0) del lugar y realizar la conversión con:

$$P_0 = P * 2^{\frac{h}{h_0}} \tag{6}$$

Donde h es la altura y P_0 la presión a nivel del mar. Si queremos ver las variables que podemos modificar, podremos mostrar por pantalla su valor introduciendo el comando 'atm /print'. Finalmente se usará 'examine' para obtener los datos de opacidad. En la imagen 11 se muestra un ejemplo de algunos de los comandos.

8.3. Python GILDAS

Es posible también crear scripts con Python con los que comandar GILDAS. Para poder llamar a GILDAS desde Python será necesario importar las bibliotecas 'pyclass' o 'pyastro'. Hay que tener en cuenta que no será posible cargar las dos bibliotecas en un mismo script.

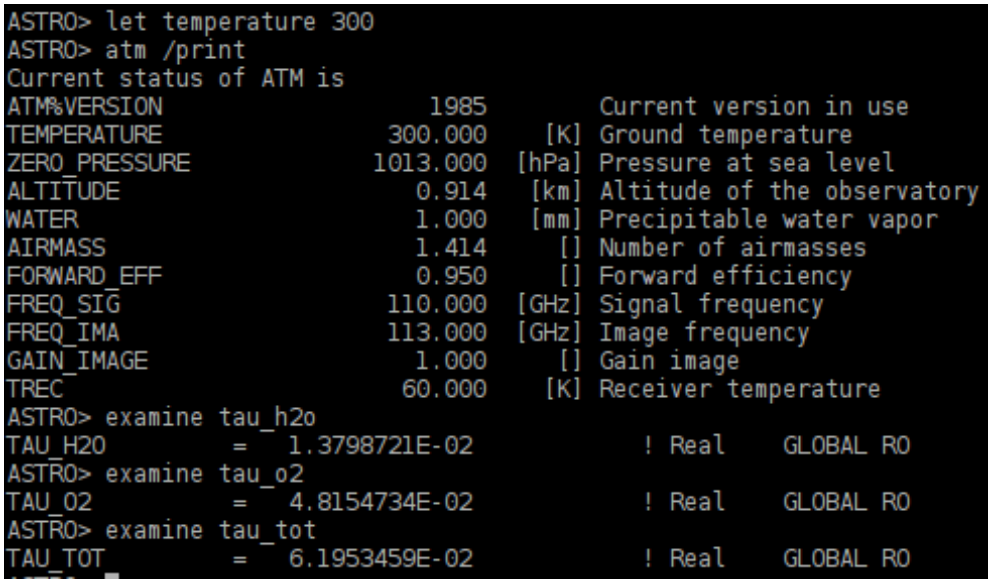

Figura 11: *Ejemplo del uso de ATM.*

Para pasarle un comando a una de estas bibliotecas con Python introduciremos, por ejemplo: '>pyastro.comm('atm /print')', o sea, pasaremos a CLASS o ASTRO el comando que queremos introducir como cadena de caracteres mediante la función 'comm'. Estas bibliotecas no permiten el uso de los bucles de GILDAS por lo que tendremos que utilizar directamente las ordenes 'for' y 'while' de Python.

Finalmente, si lo que queremos es extraer algún valor de un parámetro de CLASS cargaremos los diccionarios con los que podremos obtener cualquier variable de la lista de 'examine', por ejemplo con: '>tau=pyastro.gdict.tau_tot'.

Si se quiere conocer mejor el funcionamiento de PyGILDAS es conveniente leer los manuales de GILDAS y repasar algunos de los scripts de Python que hemos creado para el análisis de los datos con CLASS, véase el apéndice C.

Anexos

A. Datos de referencia

Banda	Frec [GHz]	Longitud de onda [mm]	HPBW [”]	Longitud del barrido [”]	Referencia [”]
S	2.5	120	2250	13500	7900
X	8.4	36	675	4000	2400
Ka	30.0	10	188	1200	700

Tabla 1: *Tabla de referencia de los valores que usamos para la longitud del barrido, referencia...*

B. Lista de fuentes de referencia.

Fuente			AR	Dec	Banda S	Banda X	Banda Ka
PER-A	3C84	J0319+415	03:19:48.16	+41:30:42.10	✓	✓	✓?
PER-B	3C123	J0437+294	04:37:04.37	+29:40:13.82	✗?	✗?	✗
TAU-A	3C144	J0531+219	05:34:31.55	+22:00:52.68	✓	✓	✓
ORIIRC2 | ORI-A	3C145	J0532-054	05:35:14.48	-05:22:30.57	✓	✓	✓
HYA-A	3C218	J0915-118	09:18:05.65	-12:05:43.99	✗	✗	✗
	3C227	J0947+072	09:47:45.15	+07:25:20.60	✗	✗	✗
VIR-A	3C274	J1230+123	12:30:49.40	+12:23:28.00	✓	✓	✗?
HER-A	3C348	J1651+046	16:51:08.20	+04:59:33.00	?	?	?
	3C353	J1720-006	17:20:28.15	-00:58:47.12	?	?	?
SGR-A		J1745-290	17:45:40.04	-29:00:28.17	?	✓?	?
CYG-A	3C405	J1959+404	19:59:28.36	+40:44:02.10	?	✓?	?
CAS-A	3C461	J2323+585	23:23:24.80	+58:48:52.31	✓	✓	✓
PLANETAS			-	-	✗	✗	✓

Tabla 2: *Fuentes preferibles para puntería en las diferentes bandas del RAEGE.*

Fuente	freq min [MHz]	freq max [MHz]	c0	c1	c2	tamaño ["]	tipo
TAU-A	500	35000	3.915	-0.299	0.0	300	gauss
VIR-A	500	10550	4.484	-0.603	-0.028	200	gauss
CYG-A	20	2000	4.695	0.085	-0.178	115	gauss
CYG-A	2000	31000	7.161	-1.244	0.0	115	gauss
DR21	4600	8900	1.29	0.0	0.0	20	gauss
DR21	21000	24000	1.245	0.0	0.0	20	gauss
CAS-A *	300	31000	5.660	-0.760	0.0	240	disco

Tabla 3: *Fuentes de flujo conocido para cálculos de eficiencia.*CAS-A tiene un flujo que disminuye con el tiempo, los parámetros en la tabla se obtuvieron en 2006.*

$$Flujo = 10^{c0+c1log(f)+c2(log(f))^2} \quad (7)$$

C. Scripts

Nombre	Uso	Función
PointFitFile.py	Todo tipo de observaciones.	Se comunica con CLASS para leer un archivo '.40m', procesar las observaciones y generar los archivos point y fit con todos los datos sobre las punterías en observaciones de continuo.
coberturaSMAzores.py	Puntería.	Dibuja la cobertura del cielo al introducir las fuentes disponibles en la observación de puntería. Este programa requiere del uso de un catálogo de fuentes, 'Catalogo.txt' y de ASTRO para coordenadas de planetas.
createPointModelFile.py	Puntería.	Combina los archivos point y fit de forma que el archivo resultante pueda ser interpretado por el programa '*OAN 40m Pointing Model*'.
getzfocusRaege.py	Foco.	Obtiene información sobre los scans de la base de datos de la antena, focos, temperaturas, tipo de scan... En el apéndice D se muestra la lista de parámetros que podemos obtener de la base de datos.
focusFile.py	Foco.	Combina los point de CLASS con los archivos generados por '*getzfocusRaege.py*' para generar un único archivo con todos los datos.
eleVStime.py	Todo tipo de observaciones.	Dibuja la posición de las fuentes en al cielo a lo largo de un día. Este programa requiere del uso de un catálogo de fuentes, 'Catalogo.txt' y de ASTRO para coordenadas de planetas.
skydip.py	Skydips.	Dibuja los datos de un skydip concreto y un ajuste de la ecuación 5 a uno o dos parámetros

Tabla 4: *Scripts que solemos usar en el procesado de los datos.*

D. Parámetros de Observations DataBase

Código	Variable	Código	Variable
Id	int(10)	XTiltFocus	double
Scan	int(5)	YTiltFocus	double
ProjectID	varchar(20)	Offsets	varchar(20)
Observer	varchar(5)	Reference	varchar(20)
Operator	varchar(5)	TypeScan	varchar(10)
DateObs	date	ModeObs	varchar(10)
UT	time	Geometry	varchar(10)
LST	time	Sub	varchar(10)
ScanEndTime	timestamp	PWV	double
Source	varchar(15)	Temperature	double
SourceVelocity	double	Pressure	double
Az	double	Humidity	double
El	double	DewPoint	double
CA	double	WinDir	double
IE	double	WinSpeed	double
XFocus	double	FEBEs	varchar(40)
YFocus	double	FEBEs	varchar(40)
ZFocus	double	Line	varchar(50)
Comments	varchar(50)	Opacity0	double
Frequency	double		

Tabla 5: *Códigos para obtener parámetros de la base de datos. Ver script 'getzfocusRaege.py'.*

E. Ecuaciones útiles

$$\eta_a = 20{,}168\frac{C_s T'_a}{S_f} \tag{8}$$

$$C_s = \begin{cases} 1+x^2 & \text{Fuente gaussiana} \\ \frac{x^2}{1-e^{-x^2}} & \text{Fuente disco} \end{cases} \tag{9}$$

$$x = \frac{\theta_s('')}{1{,}2\theta_b('')} \tag{10}$$

$$\theta_b[rad] = 1{,}2\frac{\lambda}{D} \tag{11}$$

$$T^a_{sky} = \eta_f T_{atm}(1 - e^{-\tau_0/sin(el)}) + T_{rx} + T_g(1 - \eta f) \tag{12}$$