Restart the Maple server to clear old definitions, etc.

```
> restart:
```

Load the special Maple tools packages needed for operations within the *MCMC_SPIFI* procedure.

```
> with(plots): with(plottools): with(stats): with(statplots): with
  (StringTools):
Warning, the name changecoords has been redefined
Warning, the assigned name arrow now has a global binding
Warning, the previous binding of the name transform has been
removed and it now has an assigned value
Warning, the assigned name Group now has a global binding
```

****

Initialize the procedure

****

The *MCMC_SPIFI_manualXY* procedure takes the following arguments:
Description: a string that will appear in the final spectrum title.
X: data array of velocity values (in km/s).
Y: data array of main beam temperature values (in K).
RasterPosition: the raster position of the current data, in the form: (delta "row pixels", delta "column pixels").
BadBins: a list (enclosed in square brackets and separated by commas) of bins to be removed from the spectrum before the fit is attempted. If no bins are to be removed, an empty list must be entered: [ ].
CParam: a list of the minimum, maximum, step size, and initial value to use when fitting the Lorentzian centroid.
FParam: a list of the minimum, maximum, step size, and initial value to use when fitting the Lorentzian FWHM.
OParam: a list of the minimum, maximum, step size, and initial value to use when fitting the baseline offset. In this program, I have defined the baseline as y=m(x-average(x[i]))+b, such that the offset, b, is the offset at the center of the spectrum, not at x=0.
SParam: a list of the minimum, maximum, step size, and initial value to use when fitting the baseline slope.
HParam: a list of the minimum, maximum, step size, and initial value to use when fitting the Lorentzian height.
N: total number of steps to attempt in your Markov Chain random walk.
BadN: number of steps to throw out from the beginning of the Markov Chain (start-up iterations). This is only necessary if you choose an initial point far from the chi^2 minimum.
sigma: a parameter that controls the size of proposed steps in the Markov Chain. More precisely, *sigma* is standard deviation of the random normal distribution which is used to determine the Markov step size.
r2: a parameter that controls the acceptance rate of proposed steps in the Markov Chain. However, in principle the acceptance rate is best controlled with sigma: while increasing r2 can lower the standard for acceptance, lowering sigma can raise the actual number of points with good fits to the data, which is a much more desireable means of increasing the acceptance rate.
Nvar: Number of unconstrained variables. (Degrees of Freedom (d.o.f.) = *NumBins* - *Nvar*).
RandSeed: the seed to use when generating random numbers. It can be set to either an integer or to `random`.
AdditionalError: sometimes the chi^2 of a fit is so large that the significance, alpha, is too small to be computed to the digits of precision in the program (currently Digits=50). This means that we have grosely underestimated the real error of the data, and must add some additional systematic error. This parameter allows the user to add some error (in units of K), which will add in quadrature to the previously calculated error of the data.

```
> MCMC_SPIFI_manualXY:=proc(Description,RasterPosition,X,Y,
  AvgError,BadBins,CParam,FParam,SParam,OParam,HParam,N,BadN,sigma,
  r2,Nvar,RandSeed,AdditionalError)
```

Define the local variables. These variables will only retain their definitions within the *MCMCGridSPIFI* procedure.

```
> local
  i, j, k,
  CMin, CMax, dC, C0, Cmean, Csigma, Cgrid, Cfit,
  FMin, FMax, dF, F0, Fmean, Fsigma, Fgrid, Ffit,
  SMin, SMax, dS, S0, Smean, Ssigma, Sgrid, Sfit,
  OMin, OMax, dO, O0, Omean, Osigma, Ogrid, Ofit,
```

```
  HMin, HMax, dH, H0, Hmean, Hsigma, Hgrid, Hfit,
  XBadBins, YBadBins,
  Mins, Maxs, ds,
  Mnew, Seed, Rand1, Rand2, r1, r, Naccept, AcceptRate, Nbestfit,
  pegged,
  M, L1, L2, chi1, chi2, chif, a, a1, a2, af, amax, Baseline,
  SysErrorSqd, TotError, Signal, Noise, SNR, Intensity, Isigma:
```

Define the global variables. These are variables that retain their definitions outside of the *MCMCGridSPIFI* procedure (including its sub-procedures).

```
> global
  XBins, YBins, XScaled, YScaled, XFinal, YFinal, Lf,
  graph1a, graph1b, graph2a, graph2b, graph3a, graph3b, graph3c,
  title1, title2, title3,
  AvgErrorScaled, NumBins, BinLength, BinLengthScaled, Histogram,
  Cpdf, Fpdf, Spdf, Opdf, Hpdf:
```

Rename and reorganize the fitting parameters:

```
> CMin:=CParam[1]: CMax:=CParam[2]: dC:=CParam[3]: C0:=CParam[4]:
> FMin:=FParam[1]: FMax:=FParam[2]: dF:=FParam[3]: F0:=FParam[4]:
> SMin:=SParam[1]: SMax:=SParam[2]: dS:=SParam[3]: S0:=SParam[4]:
> OMin:=OParam[1]: OMax:=OParam[2]: dO:=OParam[3]: O0:=OParam[4]:
> HMin:=HParam[1]: HMax:=HParam[2]: dH:=HParam[3]: H0:=HParam[4]:
> Mins:=[CMin,FMin,SMin,OMin,HMin]: Maxs:=[CMax,FMax,SMax,OMax,
  HMax]: ds:=[dC,dF,dS,dO,dH]:
```

Specify the number of decimal places to carry through calculations.

```
> Digits:=50:
> interface(prettyprint=0): interface(displayprecision=20):
```

****

Read and display raw data

****

Call a sub-procedure, named *HistDiscont*, that creates histogram-style displays of data. The suffix "Discont" refers to the fact that it can create histograms of discontinuous data sets (i.e. where bins have been removed).

```
> read("C://Documents and Settings/Tom Oberst/Desktop/SPIFI &
  ZEUS/SPIFI observing plan/Maple Worksheets/SPIFI/SPIFI histogram
  discontinuous v6.mpl"):
> graph1a:=pointplot({seq([X[k],Y[k]], k=1..nops(X))}):
> HistDiscont(X,Y): graph1b:=Histogram:
> title1:=cat(`[NII] 205 micron scaled data`,`\n`,`, SPIFI Car II
  Raster Position `,RasterPosition):
```

****

Scale data and display scaled data

****

Remove bins:

```
> NumBins:=nops(X)-nops(BadBins): BinLength:=(X[1]-X[nops(X)])/
  (nops(X)-1):
> for i from 1 to nops(BadBins) do XBadBins[BadBins[i]]:=DEL:
  YBadBins[BadBins[i]]:=DEL: end do:
> j:=1:
> for i from 1 to nops(X) do
> if XBadBins[i]<>DEL then XBins[j]:=X[i]: YBins[j]:=Y[i]: j:=j+1:
  end if:
> end do:
```

Scale Data (because X and Y values are entered manually already in units of K and km/s, no scaling is necessary):

```
> for i from 1 to NumBins do
> YScaled[i]:=YBins[i]:
> XScaled[i]:=XBins[i]:
> end do:
> AvgErrorScaled:=sqrt(AdditionalError^2+AvgError^2):
> BinLengthScaled:=BinLength:
```

****

Output "header"

****

User Inputs:

Description
Raster Position
X1 (km/s)
Y1 (K)
Avgerage Error "AvgError" (K)
Bins Removed; "BadBins"
Centroid Min (km/s)
Centroid Max (km/s)
Centroid s (km/s)
Centroid0 (km/s)
FWHM Min (km/s)
FWHM Max (km/s)
FWHM s (km/s)
FWHM0 (km/s)
Slope Min (K/km/s)
Slope Max (K/km/s)
Slope s (K/km/s)
Slope0 (K/km/s)
Offset Min (K)
Offset Max (K)
Offset s (K)
Offset0 (K)

Height Min (K)
Height Max (K)
Height s (K)
Height0 (K)
MCMC steps; "N"
MCMC initial steps to throw out; "BadN"
sigma of MCMC random gaussian
acceptance aggression; "r2"
unconstrained variables; "Nvar"
Random seed; "RandSeed"
Additonal User Defined Error (K); "AdditionalError"

Read-in Data:

Transmission (%)
Antenna Velocity (km/s)
Flatfielding
Gain (mV/K)
Efficiency (%)
"Flat Velocity" (counts)
HOFPI offset (counts)
HOFPI slope (km/s per count)

```
print(Description):
print(RasterPosition):
print(X[1]):
print(Y[1]):
print(AvgError):
print(BadBins):
print(CMin):
print(CMax):
print(dC):
print(C0):
print(FMin):
print(FMax):
print(dF):
print(F0):
print(SMin):
print(SMax):
print(dS):
print(S0):
print(OMin):
print(OMax):
print(dO):
print(O0):
print(HMin):
print(HMax):
```

```
print(dH):
print(H0):
print(N):
print(BadN):
print(sigma):
print(r2):
print(Nvar):
print(`see User Outputs below`):
print(AdditionalError):

print(Transmission):
print(AntennaVelocity):
print(Flatfield[Pixel]):
print(Gain):
print(Efficiency):
print(FlatVelocity[Pixel]):
print(HOFPIOffset):
print(HOFPISlope):
```

****

Markov Chain Monte Carlo

****

Calculate the figures of merit for the first point in the Markov Chain, $M_{0:}$

```
> M[0]:=[C0,F0,S0,O0,H0]:
```

Lorentzian (the baseline is defined such that the offset occurs at the spectrum's center, not at x=0):

```
> L1:=unapply(H0/(1+((x-C0)/(F0/2))^2)+(S0*(x-sum(XScaled[k],k=1..
  NumBins)/NumBins)+O0), x):
```

Chi squared:

```
> chi1:=sum((L1(XScaled[k])-YScaled[k])^2, k=1..NumBins)/
  (AvgErrorScaled^2):
```

Alpha ("confidence" or "significance level"):

```
> a1:=1-statevalf[cdf,chisquare[NumBins-Nvar]](chi1):
```

The following variables need initialized before starting the Markov Chain. Naccept is a tally of the number of times a proposed point Mnew is accepted, a is an array of the confidences of each point visited in the Markov Chain, Nbestfit is the position (i.e. iteration number) of the maximum value in a, Mnew is the first proposed step in the MCMC, and Seed is used to generate random numbers based on a user-supplied integer or your computer's clock:

```
> Naccept:=0: a[0]:=a1: Nbestfit:=0: Mnew:=[C0,F0,S0,O0,H0]:
> if type(RandSeed, integer)=true then Seed:=randomize(RandSeed):
  else Seed:=randomize(): end if:
```

Do the Markov Chain loop:

```
> for i from 1 to N do
```

Generate a new proposed point, Mnew:

```
> Rand1:=[random[normald[0,sigma]](5)]: pegged:=false:
> for j from 1 to 5 do Mnew[j]:=M[i-1][j]+ds[j]*Rand1[j]:
```

```
> if Mnew[j]>Maxs[j] then Mnew[j]:=Maxs[j]: pegged:=true:
> elif Mnew[j]<Mins[j] then Mnew[j]:=Mins[j]: pegged:=true:
> end if: end do:
```

Calculate the figures of merit for the propsed point, Mnew:

```
> L2:=unapply(Mnew[5]/(1+((x-Mnew[1])/(Mnew[2]/2))^2)+(Mnew[3]*(x-
  sum(XScaled[k],k=1..NumBins)/NumBins)+Mnew[4]), x):
> chi2:=sum((L2(XScaled[k])-YScaled[k])^2, k=1..NumBins)/
  (AvgErrorScaled^2):
> a2:=1-statevalf[cdf,chisquare[NumBins-Nvar]](chi2):
```

Compare Mnew with the last point in the Markov Chain, M[i-1], and decide whether to accept or reject it:

```
> r1:=a2/a1: r:=r1*r2:
```
print(chi1, chi2, a1, a2, r):

```
> Rand2:=evalf(rand(0..1000)()/1000):
> if Rand2 <= r then
> M[i]:=Mnew:
> a[i]:=a2:
> if a[i]>max(seq(a[k],k=0..i-1)) and pegged=false then Nbestfit:=i
  end if:
> a1:=a2:
> Naccept:=Naccept+1:
> else M[i]:=M[i-1]:
> a[i]:=a1:
> end if:
> end do:
```

Calculate the acceptance rate of your MCMC. This can be adjusted using r2 (raising r2 lowers the standard for accepting a propsed point), or, preferrably, using sigma (lowering sigma increases the number of proposed points that are good fits to the data). Emprical data suggests aiming for *AcceptRate* ~ 25% for data with >2 d.o.f. (Gregory 2005) .

```
> AcceptRate:=100*evalf(Naccept/N):
```

Calculate parameter pdf's, means, and sigmas:

```
> Cpdf:=histogram([seq(M[i][1], i=BadN..N)], area=N):
> Cmean:=describe[mean]([seq(M[i][1], i=BadN..N)]):
> Csigma:=describe[standarddeviation]([seq(M[i][1], i=BadN..N)]):

> Fpdf:=histogram([seq(M[i][2], i=BadN..N)], area=N):
> Fmean:=evalf(describe[mean]([seq(M[i][2], i=BadN..N)])):
> Fsigma:=describe[standarddeviation]([seq(M[i][2], i=BadN..N)]):

> Spdf:=histogram([seq(M[i][3], i=BadN..N)], area=N):
> Smean:=describe[mean]([seq(M[i][3], i=BadN..N)]):
> Ssigma:=describe[standarddeviation]([seq(M[i][3], i=BadN..N)]):

> Opdf:=histogram([seq(M[i][4], i=BadN..N)], area=N):
> Omean:=describe[mean]([seq(M[i][4], i=BadN..N)]):
> Osigma:=describe[standarddeviation]([seq(M[i][4], i=BadN..N)]):
```

```
> Hpdf:=histogram([seq(M[i][5], i=BadN..N)], area=N):
> Hmean:=describe[mean]([seq(M[i][5], i=BadN..N)]):
> Hsigma:=describe[standarddeviation]([seq(M[i][5], i=BadN..N)]):
```

****

Use the point in the MCMC with maximum alpha (*Nbestfit*) to do final fit

****

Compute the baseline and subtract it from the data points. (The baseline is defined such that the offset is at the center of the spectrum, not at x=0):

```
> Baseline:=unapply(M[Nbestfit][3]*(x-sum(XScaled[k],k=1..NumBins)
  /NumBins)+M[Nbestfit][4], x):
> for i from 1 to NumBins do YFinal[i]:=YScaled[i]-Baseline(XScaled
  [i]): XFinal[i]:=XScaled[i]: end do:
```

Compute the Lorentzian (without the baseline, since the baseline has now been subtracted from the data):

```
> Lf:=unapply(M[Nbestfit][5]/(1+((x-M[Nbestfit][1])/(M[Nbestfit][2]
  /2))^2), x):
> chif:=sum((Lf(XFinal[k])-YFinal[k])^2, k=1..NumBins)/
  (AvgErrorScaled^2):
> af:=1-statevalf[cdf,chisquare[NumBins-Nvar]](chif):
```

If chi^2 is not equal to 1, then there is some systematic error that has been underestimated or overestimated. The total error (TotError) is the sum of the calculated error (AvgErrorScaled) and the systematic error (SysError):

```
> SysErrorSqd:=fsolve(sum((Lf(XFinal[k])-YFinal[k])^2,k=1..NumBins)
  /(AvgErrorScaled^2+x)/(NumBins-Nvar)=1,x):
> TotError:=sqrt(AvgErrorScaled^2+SysErrorSqd):
```

Compute the Signal, Noise, SNR, Intensity, and Intensity error:

```
> Signal:=evalf((Pi/2)*M[Nbestfit][5]*(M[Nbestfit][2]
  /BinLengthScaled)):
> Noise:=evalf(sqrt((TotError^2)*((Pi/2)*M[Nbestfit][2]
  /BinLengthScaled))):
> SNR:=Signal/Noise:
> Intensity:=Signal*BinLengthScaled:
> Isigma:=Noise*BinLengthScaled:
```

Produce a graph of the final fit to the data:

```
> graph3a:=pointplot({seq([XFinal[k],YFinal[k]], k=1..NumBins)}):
> HistDiscont(convert(XFinal,list),convert(YFinal,list)): graph3b:=
  Histogram:
> graph3c:=plot(Lf(x), x=XFinal[NumBins]-BinLengthScaled/2..XFinal
  [1]+BinLengthScaled/2):
> title3:=cat(Description,`\n`,`SPIFI Raster Position `,
  RasterPosition):
```

****

Print outputs

****

fitting method
random number generator seed; "Seed"
MCMC acceptance rate (%)
final reduced chi^2; "chif/(NumBins-Nvar)"
final confidence; "af" (%)
Known Error; "AvgErrorScaled" (K)
Unknown Error Squared; "SysErrorSqd" (K^2)
Total Error; "TotError" (K)
Signal (K*Nbins)
Noise (K*sqrt(Nbins))
SNR
Intensity (K*km/s)
Intensity sigma (K*km/s)
Centroid (km/s)
Centroid sigma (km/s)
FWHM (km/s)
FWHM sigma (km/s)
Slope (K/km/s)
Slope sigma (K/km/s)
Offset (K)
Offset sigma (K)
Height (K)
Height sigma (K)
Height sigma alternative (non-Gauss pdf) (K)
Final list of X values (km/s)
Final list of Y values (K)

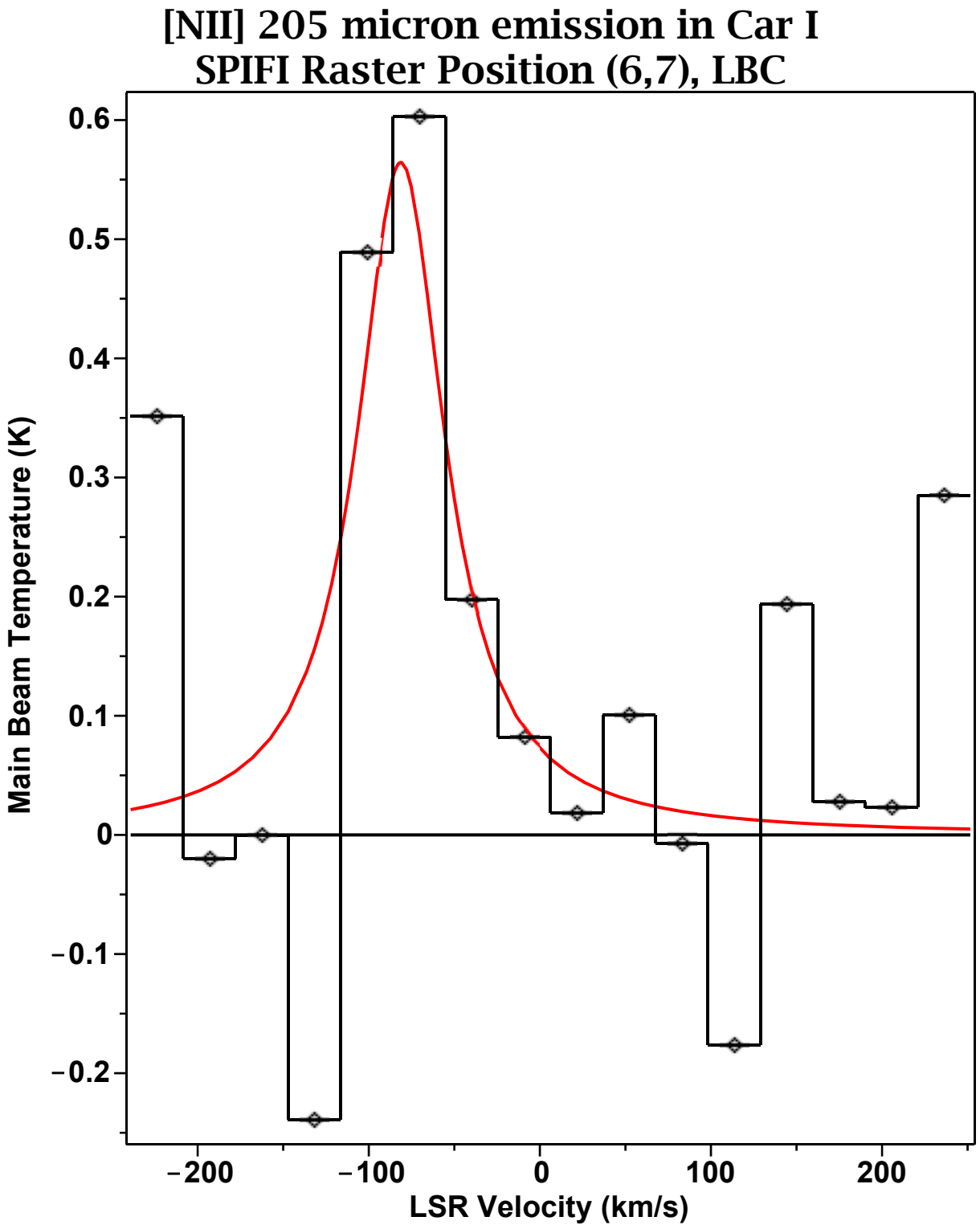
```
> print(`MCMC min fit`):
> print(Seed):
> print(AcceptRate):
> print(chif/(NumBins-Nvar)):
> print(100*af):
> print(AvgErrorScaled):
> print(SysErrorSqd):
> print(TotError):
> print(Signal):
> print(Noise):
> print(SNR):
> print(Intensity):
> print(Isigma):
> print(M[Nbestfit][1]):
> print(Csigma):
> print(M[Nbestfit][2]):
> print(Fsigma):
> print(M[Nbestfit][3]):
> print(Ssigma):
> print(M[Nbestfit][4]):
```

```
> print(Osigma):
> print(M[Nbestfit][5]):
> print(Hsigma):
> print(TotError*sqrt(chif/(NumBins-Nvar))):
> for i from 1 to nops(X) do print(XFinal[i]) od:
> for i from 1 to nops(X) do print(YFinal[i]) od:

****************************************

> end proc:
> MCMC_SPIFI_manualXY(`[NII] 205 micron emission in Car I`,`(6,7),
  LBC`,[236.259,205.586,174.914,144.242,113.570,82.898,52.226,
  21.553,-9.119,-39.791,-70.463,-101.135,-131.808,-162.480,
  -193.152,-223.824],[0.1389222,-0.1103345,-0.0928093,0.0859553,
  -0.2711668,-0.0895290,0.0311870,-0.0381949,0.0382353,0.1664311,
  0.5845062,0.4836142,-0.2317817,0.0199565,0.0130529,0.3970861],
  0.205363363,[],[-200,150,15,-30],[61,185,15,90],[-0.005,-0.00025,
  5.0*10^(-4),-3.0*10^(-4)],[-6,6,.08,0.0],[0,2,.15,.35],5000,0,1,
  1,5,random,0):
`[NII] 205 micron emission in Car I`
`(6,7), LBC`
236.259
.1389222
.205363363
[]
-200
150
15
-30
61
185
15
90
-0.5e-2
-0.25e-3
0.5000000000e-3
-0.3000000000e-3
-6
6
0.8e-1
0.
0
2
```

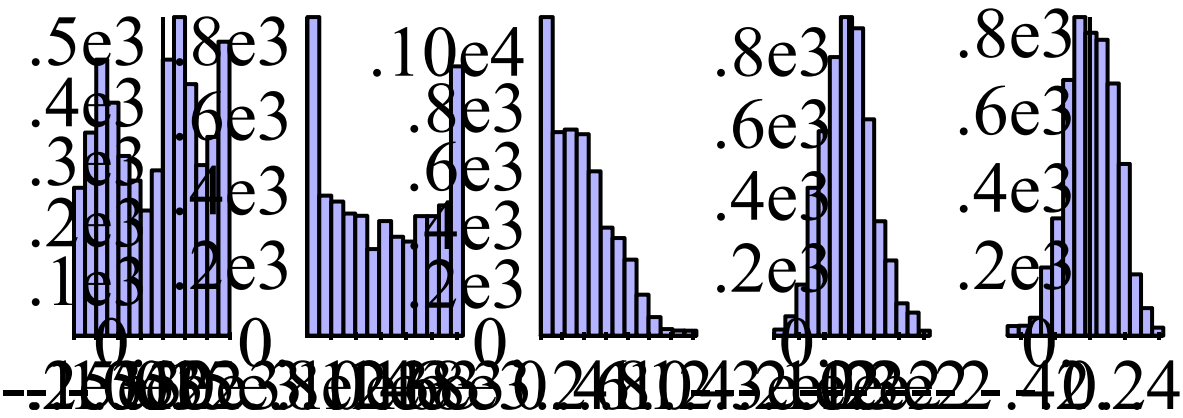
```
.15
.35
5000
0
1
1
5
`see User Outputs below`
0
Transmission
AntennaVelocity
Flatfield[Pixel]
Gain
Efficiency
FlatVelocity[Pixel]
HOFPIOffset
HOFPISlope

`MCMC min fit`
1211239906
44.800000000000000000000000000000000000000000000000
.956136170771283153697198405886284494821715978697 09
48.452403075936609378333787374188226278190789966013
.205363363000000000000000000000000000000000000000000
-0.184991799675311886302005770477354547100523450954 1e-2
.200808846582805335670615534061401874471085578880 58
1.804766538651224995176321908615859624107943960453 7
.359069688086097574346662540679405607496661799652 29
5.026229165349314743407958385362454207285906738610 3
55.356160226818103297047180845447369562563678743828
11.013457286914402019875702780226864674259110051295
-81.218541433134101116544933210214316616116463679697
88.703704095722114141196197820780096762270173872787
62.433381251550529805454977270377925514143937144335
40.170082432027884343674562032618449220744979658783
-0.416934245587736551995379100998669003782992195983 66e-3
0.348884461727449698690082245073087557486742932860 63e-3
-0.502481435095604444088217752903662178057305369514 4e-1
0.896051907698535190780393138422154420062200197125 48e-1
.564454870396196855504016268126741181586692614278 33
.261796330702952783596770891974379339809783452774 32
.196355339515533012268502548311046379431343678816 07
```

```
236.259
205.586
174.914
144.242
113.570
82.898
52.226
21.553
-9.119
-39.791
-70.463
-101.135
-131.808
-162.480
-193.152
-223.824
.285082626999493139368804775597526783106726681951860
0.230373028845804961094505124325946087536909623244 6e-1
0.277742957039134405866482446467634330696590256892 5e-1
.193750688523246385063845976860932257385627089054040
-.176159618657420670458956290924898918298404847581180
-0.731002583808772598175855871073009398243678421638 6e-2
.100617766981245218495439173503438730333531279148400
0.184472428663325752360849103385065559804955595209970e-1
0.820892356856655197132826425526753802964636228857860e-1
.197496828504998464190480374766844204612431686250580
.602783721324331408667678106981013028928399749615360
.489103514143664353144875839195181853244367812980150
-.239081009971248290114478423969750321108667906647250
-0.131017151915345637280691755581496792699843282464e-3
-0.198228243325824011600829595414126724767317799176750e-1
.351422168486750543317114772672756151839236283447110
```

```
> display(graph3a,{op(graph3b),graph3c,plot(0, x=XFinal[1]+
  BinLengthScaled/2..XFinal[NumBins]-BinLengthScaled/2,color=black)}, axes=
  boxed, title=title3, titlefont=[Times,14,bold], labels=["LSR Velocity
  (km/s)","Main Beam Temperature (K)"], labelfont=[Arial,11,bold], font=[Arial,
  11,bold], labeldirections=[horizontal,vertical]);
```



```
> display(array([Cpdf,Fpdf,Hpdf,Spdf,Opdf]));
```




```
> display(graph1a,{op(graph1b)});
```