Restart the Maple server to clear old definitions, etc.

```
> restart:
```

Load the special Maple tools packages needed for operations within the *MCMCGridSPIFI* 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_TREND* procedure takes the following arguments:
Path: the file path where the TREND LO data files (*.spi) and other files accessed by *MCMC_TREND* are stored.
File: the filename of the data set you wish to work with.
Pixel: the pixel whose spectrum you wish to fit.
BadBins: a list (enclosed in square brackets and separated by commas) of bins to be removed from the spectrum before the fit is attempted. SPIFI bins (for our best TREND LO measurements) are numbered 1-32 from right to left. If no bins are to be removed, an empty list must be entered: [ ].
CParam: a list of the maximum, minimum, step size, and initial value to use when fitting the Lorentzian centroid.
FParam: a list of the maximum, minimum, step size, and initial value to use when fitting the Lorentzian FWHM.
OParam: a list of the maximum, minimum, 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 the y-axis (i.e., not at x=0).
SParam: a list of the maximum, minimum, step size, and initial value to use when fitting the baseline slope.
HParam: a list of the maximum, minimum, 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. (In principle this is best done by changing sigma. While r2 lowers the standard on what points are accepted, sigma raises the actual number of points with good fits to the data!).
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 so small that it cannot 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_TREND:=proc(Path,File,Pixel,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,
```

```
  ErrorArray, Flatfield, Gain, Efficiency, VelocityShift,
  HOFPIOffset, HOFPISlope,
  XBadBins, YBadBins,
  Mins, Maxs, ds,
  Mnew, Seed, Rand1, Rand2, r1, r, Naccept, AcceptRate, Nbestfit,
  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
  X, Y, XBins, YBins, XScaled, YScaled, XFinal, YFinal, Lf,
  graph1a, graph1b, graph2a, graph2b, graph3a, graph3b, graph3c,
  title1, title2, title3, title4, title5,
  AvgError, AvgErrorScaled, Transmission, 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(displayprecision=20):
```

****
Read and display raw data
****

Call and execute a sub-procedure, named *readSPIFIdata*, that reads SPIFI data from the *.spi file specified by the *Path*, *Filename*, and *Pixel* variables. *.spi files for SPIFI measurements of the TREND LO contain a 32x26 array of values. There is one row for each of the 32 bins (HOFPI positions) in the spectrum. The first column contains the x-axis values, the next 25 columns contain the y-values of each of the 25 pixels. This sub-procedure creates two global variables, X[i] and Y[i], each are 1D arrays with 32 values representing the X and Y coordinates of the 32 bins comprising the specified *Pixel*'s spectrum.

```
> read("C://Documents and Settings/Tom Oberst/Desktop/SPIFI &
ZEUS/SPIFI observing plan/Maple Worksheets/SPIFI/read
  SPIFI data +X v2.mpl"):
> readSPIFIdata_andX(Path, File, Pixel): X:=convert(X,list): Y:=
  convert(Y,list):
```

Define the average error for the TREND LO measurements. Because we do not coadd scans (the trend line shows up in our first scan), we do not have a standard deviation (over the coadd) as we do for long SPIFI integrations. Therefore, I define values here based on by-hand calculations of the RMS of the baselines of each pixel (the first six and last six bins in the scan) in 200LO_scan8_FFT.spi:

```
> ErrorArray:=[0.364452489,1.926014625,1.063975561,0.623923749,
  0.53178469,0.270370022,10.05585853,2.812945868,1.957531705,
  0.820739381,0.221107585,0.489243812,0.667572189,0.518265738,
  1.112308247,1.007240923,2.015210607,0.534356329,2.264868915,
  1.503586806,1.057749241,9.592102991,0.997789014,1.173887817,
  0.525737709]:
> AvgError:=ErrorArray[Pixel]:
```

Define the transmission. Since we don't go through the atmosphere for TREND LO measurements, we take this as 100%:

```
> Transmission:=100:
```

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 v4.mpl"):
```

Create a graph of raw data:

```
> graph1a:=pointplot({seq([X[k],Y[k]], k=1..nops(X))}):
> HistDiscont(X,Y): graph1b:=Histogram:
> title1:=cat(File,", pixel ",Pixel,", raw TREND LO data"):
```

****

Scale data and display scaled data

****

Y-scaling factors (flatfielding, gain, efficiency, and transmission) & X-scaling factors (velocity shift, HOFPI offset, HOFPI slope):

Define the (multiplicative) flatfield array, normalized to pixel 8:

```
> Flatfield:=[2.5, 1.13, 0.9382, 1.884, 1.731, 2.695, -16.52, 1,
  2.482, 1.320, 2.577, 2.298, 2.603, 2.772, 1.096, 1.220, 2.184,
  1.603, 1.249, 1.934, 0.7632, 11, 0.9508, 2.265, 2.611]:
```

Define the gain, based on hot & cold loads in telescope beam tube (mV/K):

```
> Gain:=5.86:
```

Define the (telescope and beam-coupling) efficiency. Since we pick up the TREND LO power in the same place that we held our hot and cold loads, we take this as 100%:

```
> Efficiency:=1:
```

Define the (additive) velocity shift array. Because this is one of the things we are determining by fitting the TREND LO lines, we set this equal to 0 here:

```
> VelocityShift:=0:
```

Define the conversion factors from HOFPI counts to km/s (the offset is in [counts], the slope is in [km/s per count]). Again, we are determining these here, so we set them to zero and unity, respectively:

```
> HOFPIOffset:=0: HOFPISlope:=1:
```

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:

```
> for i from 1 to NumBins do
> YScaled[i]:=Flatfield[Pixel]*(1/Gain)*(1/Efficiency)*
  (100/Transmission)*YBins[i]:
> XScaled[i]:=XBins[i]:
> end do:
> AvgErrorScaled:=sqrt(AdditionalError^2+(Flatfield[Pixel]*(1/Gain)
  *(1/Efficiency)*(100/Transmission)*AvgError)^2):
> BinLengthScaled:=BinLength:
```

Create a graph of the scaled data:

```
> graph2a:=pointplot({seq([XScaled[k],YScaled[k]], k=1..NumBins)}):
> HistDiscont(convert(XScaled,list),convert(YScaled,list)):
  graph2b:=Histogram:
> title2:=cat(File,", pixel ",Pixel,", scaled TREND LO data"):
```

****
Output "header"
****

Fitting program:
File
Pixel
Bins Removed; "BadBins"
HOFPI offset (counts)
HOFPI slope (km/s per count)
Velocity offset (counts)
Gain
Flatfielding
Efficiency (fraction)
Transmission (%)
Centroid Min (counts)
Centroid Max (counts)
Centroid s (counts)
Centroid0 (counts)
FWHM Min (counts)
FWHM Max (counts)
FWHM s (counts)
FWHM0 (counts)
Height Min (K)
Height Max (K)
Height s (K)
Height0 (K)

Slope Min (K/counts)
Slope Max (K/counts)
Slope s (K/counts)
Slope0 (K/counts)
Offset Min (K)
Offset Max (K)
Offset s (K)
Offset0 (K)
MCMC steps; "N"
MCMC initial steps to throw out; "BadN"
s of MCMC random gaussian
acceptance aggression; "r2"
unconstrained variables; "Nvar"
additional error; "AdditionalError" (K)

```
print(File):
print(Pixel):
print(BadBins):
print(HOFPIOffset):
print(HOFPISlope):
print(VelocityShift):
print(Gain):
print(Flatfield[Pixel]):
print(Efficiency):
print(Transmission):
print(CMin):
print(CMax):
print(dC):
print(C0):
print(FMin):
print(FMax):
print(dF):
print(F0):
print(HMin):
print(HMax):
print(dH):
print(H0):
print(SMin):
print(SMax):
print(dS):
print(S0):
print(OMin):
print(OMax):
print(dO):
print(O0):
```

```
print(N):
print(BadN):
print(sigma):
print(r2):
print(Nvar):
print(AdditionalError):
```

****

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 has a funny form so that the offset occurs at the emission line 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)]:
> 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]:
> elif Mnew[j]<Mins[j] then Mnew[j]:=Mins[j]:
> 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:
> 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)) 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. 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. Note that the baseline is defined in such a way that the offet is the offset at the center of the spectrum, not at the y-axis (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):

```
> 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)/27=1,x):
> TotError:=sqrt(AvgErrorScaled^2+SysErrorSqd):

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

> 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(File,", pixel ",Pixel,`\n`,"MCMC chi^2 fit to
  Lorentzian"):
```

****
Print outputs
****

fitting method
random number generator seed; "Seed"
MCMC acceptance rate (%)
reduced chi^2
confidence; "alpha" (%)
Known Error "AvgErrorScaled" (K)
Unknown Error Squared "SysErrorSqd" (K^2)
Total Error "TotError" (K)
Signal (K*Nbins)
Noise (K*sqrt(Nbins))
SNR
Intensity (K*counts)
Intensity s (K*counts)
Centroid (counts)
Centroid s (counts)
FWHM (counts)

FWHM s (counts)
Height (K)
Height s (K)
Slope (K/counts)
Slope s (K/counts)
Offset (K)
Offset s (K)
Final list of X values (counts)
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][5]):
> print(AvgErrorScaled*sqrt(chif/(NumBins-Nvar))):
> print(M[Nbestfit][3]):
> print(Ssigma):
> print(M[Nbestfit][4]):
> print(Osigma):
> 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_TREND("C:/Documents and Settings/Tom Oberst/Desktop/SPIFI &
ZEUS/SPIFI observing plan/Observing Data/Original Data
Files/200LO/","200LO_scan9_FFT.spi",25,[],[27525,27650,1,27625],
[0,80,3,30],[-0.1,0.1,0.0015,0.008],[0,80,0.25,40],[0,50,.25,11],
```

```
1000,0,.8,1,5,random,.5):
```

"200LO_scan9_FFT.spi"

25

[ ]

0

1

0

5.86

2.611

1

100

27525

27650

1

27625

0

80

3

30

0

50

0.25

11

-0.1

0.1

0.0015

0.008

0

80

0.25

40

1000

0

0.8

1

5

0.5

MCMC min fit

1183131827

29.00000000000000000000000000000000000000000

0.93957050864157540282172413646424503266894647785144

55.382437800924773433108364188349376077630016184500

0.55215283695099225677183477274280303590949882456904

−0.018423305535024964728632115531707522171206369593 56

0.53520972507793996115344977032170606437115900967967

304.65214368765121421912606025751296536246983267793

2.5291951143393498023460732663956767246972567533407

120.45418795901371069289437158883397636949777528183

609.30428737530242843825212051502593072493966535586

5.0583902286786996046921465327913534493945135066814

27622.927573743593670963946704424753851702561819015

0.61408457577818653179086728282787391326013029015026

30.261985440665255761565511578732501280531429962754

2.5899204045939368456182900352402487854695822993670

12.817901769598142259733236260421404148189314235125

0.53520972507793996115344977032170606437115900967966

−0.0011581778517899847279306585632326885494438808498897

0.0052003301239791825350228400871694608210644372539571

38.687921381258623724837406424048899893437860005246

0.37048589326409489376215606490216384297771914152058

27632.000

27630.000

27628.000

27626.000

27624.000

27622.000

27620.000

27618.000

27616.000

27614.000

27612.000

27610.000

27608.000

27606.000

27604.000

27602.000

27600.000

27598.000

27596.000

27594.000

27592.000

27590.000

27588.000

27586.000

27584.000

27582.000

27580.000

27578.000

27576.000

27574.000

27572.000

27570.000

8.905328731122975016745508837827695704154627263557

9.957049906136118596777702128209762750263930628137

11.253639215961548866229690640434833209342517269168

12.631205170838173674930825910339084555793117562076

13.072593956772818961447660838946407574598666660444

12.769317522570945476633437747075914893745512687140

11.822090960382723868952320457253203782858229021004

11.031350628569928882431612723744485846032378801966

10.068106071501161199665171201839863472346460323543

9.716802386446045394031835481983022668626412152286

9.064467332790246994541844472023792786408070465672

8.230039535107144840785641857968999764257988164724

7.471451039471824256995309551081442236988451938860

5.991580477283602649314192261258731126101168272723

5.122219192740432236172324118193699196101256620238

4.057411191303063870812025940998974433336839848299

3.4233117002752518194449018252376967695485322913 78

2.764684403957337378999279415961060743985480707153

2.495637021291300071659459054465994684292736290162

2.165833539990450477630218897749085621187022589896

1.834250033945573579846712529428080994941377149016

1.876250724829024327114400700356223126374912595507

1.645678610422372685303590577769006896033704014695

1.543687552500362681718036427878036399480891338319

1.520478295602243463115417431570683650368351699486

1.470238950820165200144197752669474245965709671575

1.592077427369144957650793773426967913235422592469

1.587466421494916524031109930703232911563155991178

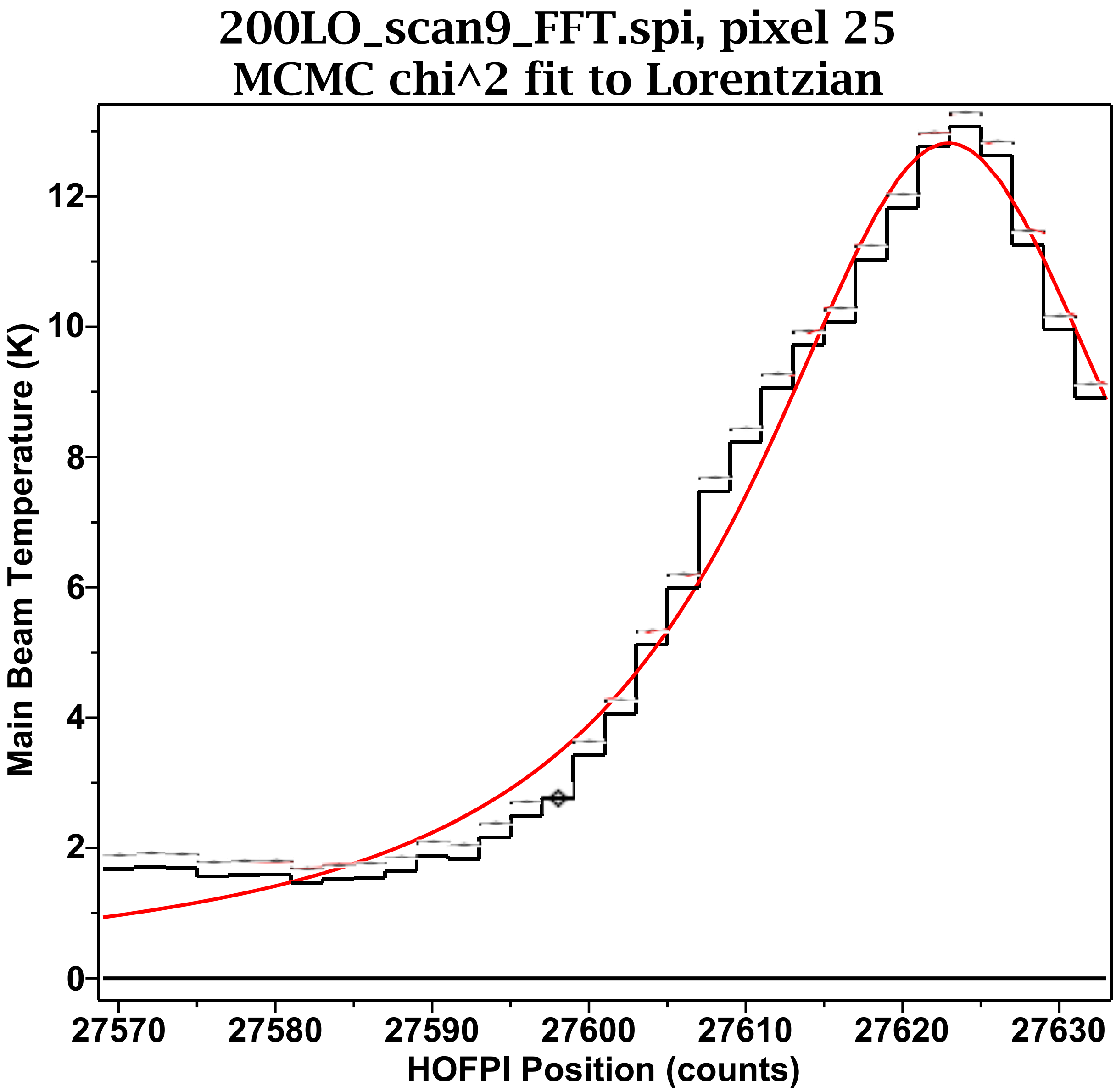
1.562384908282804131367057487296904053235599287499

1.690724151043387984436793439795012054976301969486

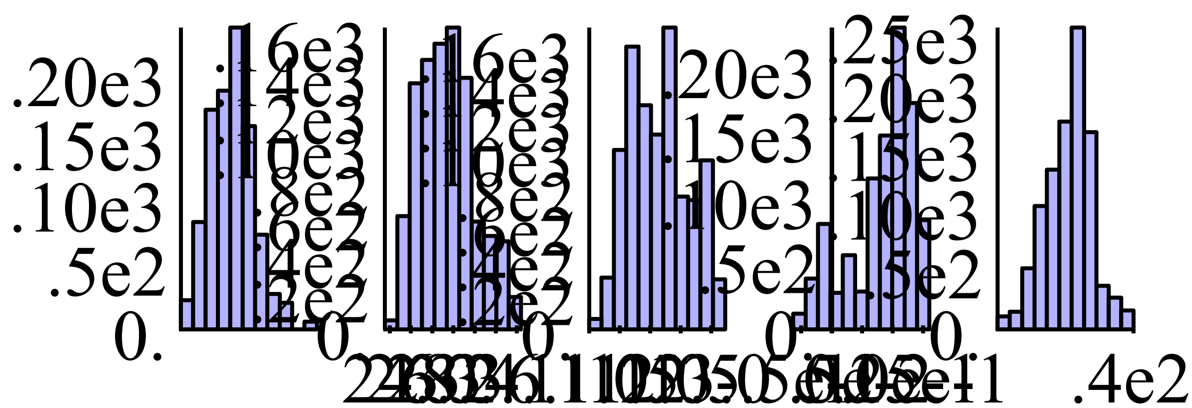
1.709164354213528151499703453726567155693113866489

1.678673258919504496037016198033548877570335319806

```
> 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=["HOFPI Position
  (counts)","Main Beam Temperature (K)"], labelfont=[Arial,11,bold], font=
  [Arial,11,bold], labeldirections=[horizontal,vertical]);
```

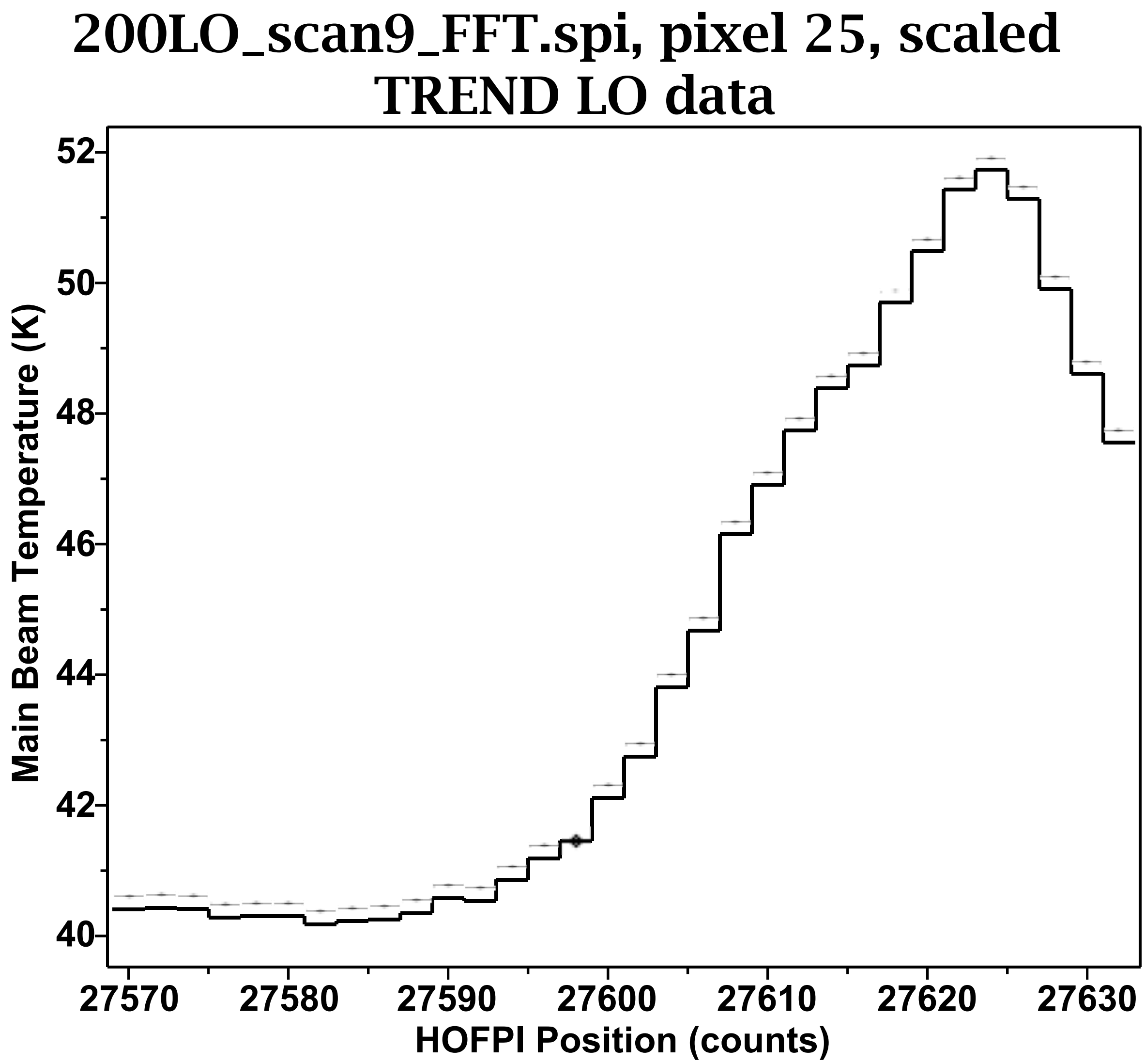



```
> display(array([Cpdf,Fpdf,Hpdf,Spdf,Opdf]));
```

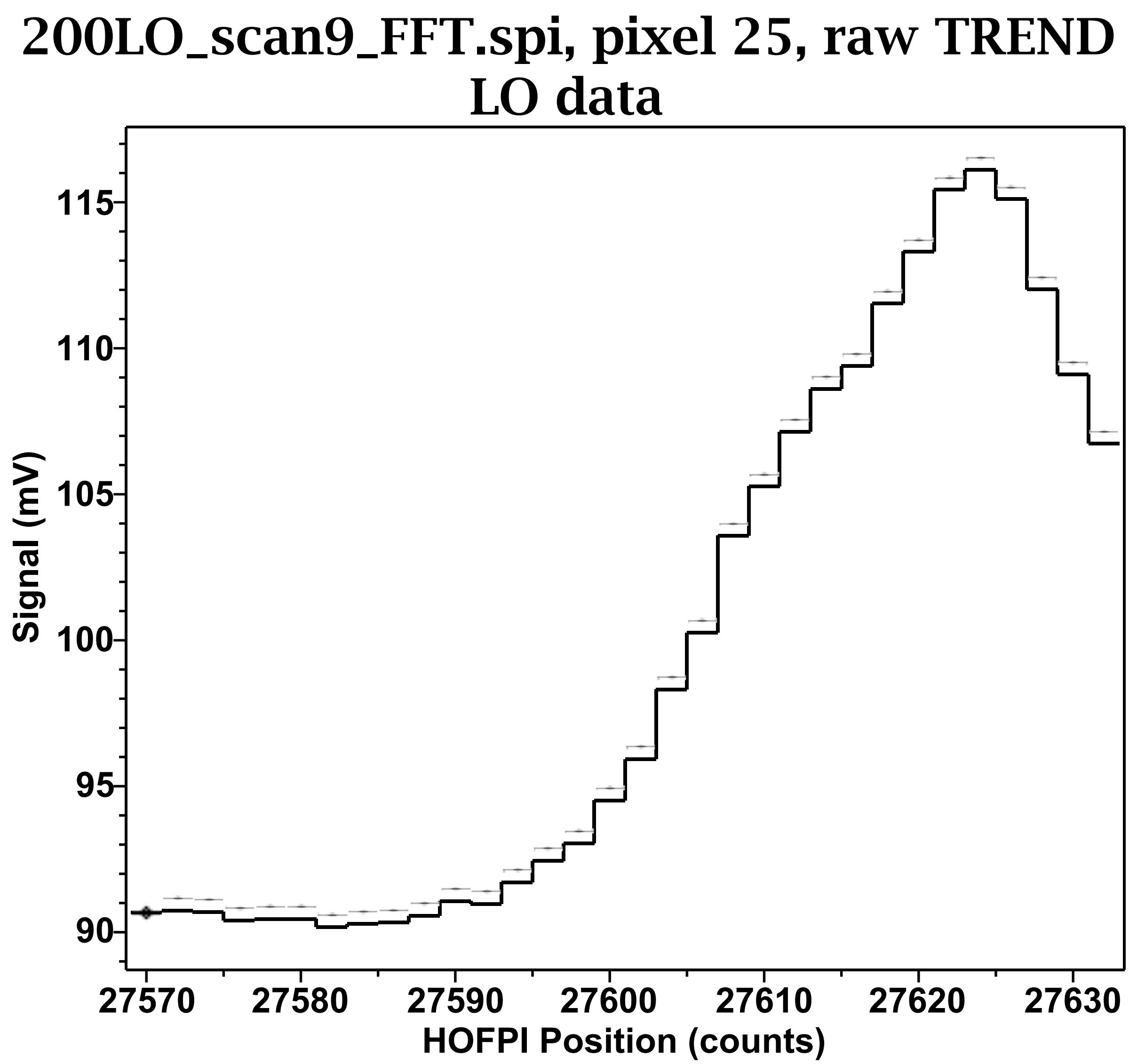


```
> display({graph2a,op(graph2b)},view=[XScaled[NumBins]
```

```
-BinLengthScaled/2..XScaled[1]+BinLengthScaled/2,(min(op(convert
(YScaled,list)))-0.05*(max(op(convert(YScaled,list)))-min(op
(convert(YScaled,list))))..(max(op(convert(YScaled,list)))+0.05*
(max(op(convert(YScaled,list)))-min(op(convert(YScaled,list)))))
],axes=boxed,title=title2, titlefont=[Times,16,bold], labels=["HOFPI
Position (counts)","Main Beam Temperature (K)"], labelfont=[Arial,11,bold],
font=[Arial,11,bold], labeldirections=[horizontal,vertical]);
```




```
> display({graph1a,op(graph1b)},view=[X[32]-BinLength/2..X[1]+
BinLength/2,(min(op(convert(Y,list)))-0.05*(max(op(convert(Y,
list)))-min(op(convert(Y,list))))..(max(op(convert(Y,list)))
+0.05*(max(op(convert(Y,list)))-min(op(convert(Y,list)))))],axes=
boxed,title=title1, titlefont=[Times,16,bold], labels=["HOFPI Position
(counts)","Signal (mV)"], labelfont=[Arial,11,bold], font=[Arial,11,bold],
labeldirections=[horizontal,vertical]);
```


200LO_scan9_FFT.spi, pixel 25, raw TREND
LO data
115
110
105
100
95
90
Signal (mV)
27570
27580
27590
27600
27610
27620
27630
HOFPI Position (counts)