Restart the Maple server to clear old definitions, etc.

```
> restart:
```

Load the special Maple tools packages needed for operations within the *MCMC_ISO* 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_ISO* procedure takes the following arguments:
Path: the file path where the ISO data files (*.spi) are stored.
File: the filename of the raster you wish to work with.
Position: the position in the raster whose spectrum you wish to fit. The positions count consecutively up from 1, in the order in which they were observed in the raster.
BadBins: a list (enclosed in square brackets and separated by commas) of bins to be removed from the spectrum before the fit is attempted. ISO bins are numbered 1-?? 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 Gaussian centroid.
FParam: a list of the maximum, minimum, step size, and initial value to use when fitting the Gaussian 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 Gaussian 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 the 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 value 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 W m^-2 um^-1), which will add in quadrature to the previously calculated error of the data.

```
> MCMC_ISO:=proc(Path,File,Position,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 *MCMC_ISO* 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, G1, G2, chi1, chi2, chif, a, a1, a2, af, amax,
  Baseline, SysErrorSqd, Signal, Noise, SNR, Intensity, Isigma:
```

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

```
> global
  X, Y, XBins, YBins, XScaled, YScaled, XFinal, YFinal, Gf, TotError,

  graph1a, graph1b, graph2a, graph2b, graph3a, graph3b, graph3c,
  title1, title2, title3,
  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, and the properties of the output display:

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

****

Read and display raw data

****

Call and execute a sub-procedure, named *readSPIFIdata_andX*, that reads ISO data from the *.spi file specified by the *Path*, *Filename*, and *Position* variables:

```
> 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, Position): X:=convert(X,list): X:=
  ListTools[Reverse](X): Y:=convert(Y,list): Y:=ListTools[Reverse]
  (Y):
```

Call and execute a sub-procedure, named *readSPIFIerrors_andX*, that reads ISO errors from the "error" version of the *.spi data file. The result is the global variable *AvgError*, which is a single value representing the average error (in flux units) of the requested *Position*'s spectrum. It is the average (across all bins in the spectrum) of the individual errors in each bin:

```
> read("C://Documents and Settings/Tom Oberst/Desktop/SPIFI &
  ZEUS/SPIFI observing plan/Maple Worksheets/SPIFI/read
  SPIFI errors +X v2.mpl"):
> readSPIFIerrors_andX(Path, File, Position):
```

Define the transmission. ISO was in space, therefore transmission is 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,", position ",Position,", raw ISO data"):
```

****

Scale data and display scaled data

****

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

Since our ISO data has already been processed, we set all of these values to zero or unity; the exception is the Gain, which I use to convert the ISO fluxes from W cm^-2 um^-1 to W m^-2 um^-1:

```
> Flatfield:=1:
> Gain:=100^2:
> Efficiency:=1:
> VelocityShift:=0:
> 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 (since the ISO data I am using is supposed to be already processed, scaling is not necessary):

```
> for i from 1 to NumBins do
> YScaled[i]:=Gain*YBins[i]:
> XScaled[i]:=XBins[i]:
```

```
> end do:
> AvgErrorScaled:=sqrt(AdditionalError^2+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,", position ",Position,", scaled ISO data"):
```

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

File
Position
Bins Removed; "BadBins"
HOFPI offset (counts)
HOFPI slope (km/s per count)
Velocity offset (counts)
Gain
Flatfielding
Efficiency (fraction)
Transmission (%)
Centroid Min (um)
Centroid Max (um)
Centroid s (um)
Centroid0 (um)
FWHM Min (um)
FWHM Max (um)
FWHM s (um)
FWHM0 (um)
Height Min (W m^-2 um^-1)
Height Max (W m^-2 um^-1)
Height s (W m^-2 um^-1)
Height0 (W m^-2 um^-1)
Slope Min (W m^-2 um^-2)
Slope Max (W m^-2 um^-2)
Slope s (W m^-2 um^-2)
Slope0 (W m^-2 um^-2)
Offset Min (W m^-2 um^-1)
Offset Max (W m^-2 um^-1)
Offset s (W m^-2 um^-1)
Offset0 (W m^-2 um^-1)
MCMC steps; "N"
MCMC initial steps to throw out; "BadN"
s of MCMC random gaussian
acceptance aggression; "r2"
unconstrained variables; "Nvar"
additional error; "AdditionalError" (W m^-1 um^-1)

```
print(File):
print(Position):
print(BadBins):
print(HOFPIOffset):
```

```
print(HOFPISlope):
print(VelocityShift):
print(Gain):
print(Flatfield):
print(Efficiency):
print(Transmission):
print(evalf(CMin)):
print(evalf(CMax)):
print(evalf(dC)):
print(evalf(C0)):
print(evalf(FMin)):
print(evalf(FMax)):
print(evalf(dF)):
print(evalf(F0)):
print(evalf(HMin)):
print(evalf(HMax)):
print(evalf(dH)):
print(evalf(H0)):
print(evalf(SMin)):
print(evalf(SMax)):
print(evalf(dS)):
print(evalf(S0)):
print(evalf(OMin)):
print(evalf(OMax)):
print(evalf(dO)):
print(evalf(O0)):
print(N):
print(BadN):
print(sigma):
print(r2):
print(Nvar):
print(evalf(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]:
```

Gaussian (the baseline has a funny form so that the offset is defined at the emission line center, not at x=0):

```
> G1:=unapply(H0*exp(-(x-C0)^2/(F0/sqrt(4*ln(2)))^2)+(S0*(x-sum
  (XScaled[k],k=1..NumBins)/NumBins)+O0), x):
```

Chi squared:

```
> chi1:=sum((G1(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:
```
> G2:=unapply(Mnew[5]*exp(-(x-Mnew[1])^2/(Mnew[2]/sqrt(4*ln(2)))^2)
  +(Mnew[3]*(x-sum(XScaled[k],k=1..NumBins)/NumBins)+Mnew[4]), x):
> chi2:=sum((G2(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(chi2, a2, a1, r):
```
> Rand2:=evalf(rand(0..1000)()/1000):
> if Rand2 <= r then
> M[i]:=Mnew:
> a[i]:=a2:
```
print(chi2/(NumBins-Nvar)):
```
> 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 sigma and r2 (in that order) . 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 Gaussian (without the baseline):

```
> Gf:=unapply(M[Nbestfit][5]*exp(-(x-M[Nbestfit][1])^2/(M[Nbestfit]
  [2]/sqrt(4*ln(2)))^2), x):
> chif:=sum((Gf(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((Gf(XFinal[k])-YFinal[k])^2, k=1..
  NumBins)/(AvgErrorScaled^2+x)/(NumBins-Nvar)=1,x):
> TotError:=sqrt(AvgErrorScaled^2+SysErrorSqd):

> Signal:=evalf(sqrt(Pi/(4*ln(2)))*M[Nbestfit][5]*M[Nbestfit][2]
  /BinLengthScaled):
> Noise:=evalf(sqrt((TotError^2)*(sqrt(Pi/(4*ln(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(Gf(x), x=XFinal[NumBins]-BinLengthScaled/2..XFinal
  [1]+BinLengthScaled/2):
> title3:=cat(File,", position ",Position,`\n`,"MCMC chi^2 fit to
  Gaussian"):

************************************
Print outputs
************************************

fitting method
random number generator seed; "Seed"
MCMC acceptance rate (%)
reduced chi^2
confidence; "alpha" (%)
Known Error "AvgErrorScaled" (W m^-2 um^-1)
Unknown Error Squared "SysErrorSqd" ((W m^-2 um^-1)^2)
Total Error "TotError" (W m^-2 um^-1)
Signal ((W m^-2 um^-1)*Nbins)
Noise ((W m^-2 um^-1)*sqrt(Nbins))
SNR
Intensity (W m^-2 )
Intensity s (W m^-2)
Centroid (um)
Centroid s (um)
FWHM (um)
FWHM s (um)
Height (W m^-2 um^-1)
Height s (W m^-2 um^-1)
Slope (W m^-2 um^-2)
Slope s (W m^-2 um^-2)
Offset (W m^-2 um^-1)
Offset s (W m^-2 um^-1)
Final list of X values (um)
Final list of Y values (W m^-2 um^-1)

> 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_ISO("C://Documents and Settings/Tom Oberst/Desktop/SPIFI &
 ZEUS/SPIFI observing plan/ISO Carina data/",
  "ISO145_Car2.spi",14,[],[145,146,.292,145.525],[.584,.85,.292,
  .8],[-20*10^(-15),20*10^(-15),2*10^(-15),2*10^(-15)],
  [-10*10^(-14),150*10^(-14),0.5*10^(-15),4.4*10^(-14)],[0.1*10^
  (-15),20*10^(-15),0.1*10^(-15),1.75*10^(-15)],10000,0,1,1,4,
  random,2*10^(-15)):




"ISO145_Car2.spi"
14
[]
0
1
0
10000
1
1
100
```

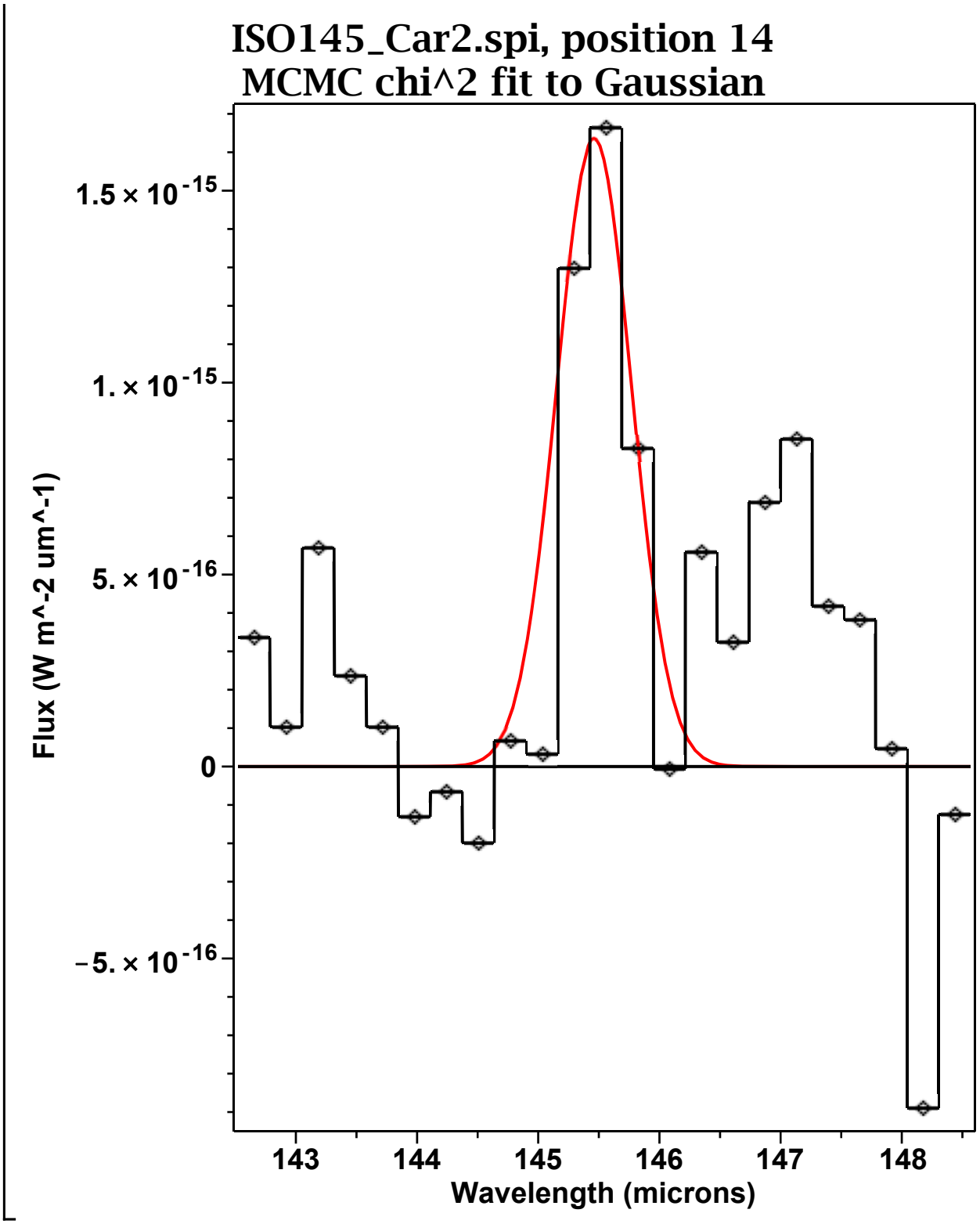
```
145.
146.
.292
145.525
.584
.85
.292
.8
0.1000000000e-15
0.200000000000000000000000000000000000000000000000e-13
0.1000000000e-15
0.1750000000e-14
-0.200000000000000000000000000000000000000000000000e-13
0.200000000000000000000000000000000000000000000000e-13
0.200000000000000000000000000000000000000000000000e-14
0.200000000000000000000000000000000000000000000000e-14
-0.100000000000000000000000000000000000000000000000e-12
0.150000000000000000000000000000000000000000000000e-11
0.5000000000e-15
0.4400000000e-13
10000
0
1
1
4
0.200000000000000000000000000000000000000000000000e-14

`MCMC min fit`
1218126566
28.780000000000000000000000000000000000000000000000
0.489251328413402963622006550374711867613256363775 31e-1
99.999999599760863823036689941914412467790557218 97
0.200000000044673417178375553813487749362768261773 37e-14
-0.380429947033414938707888139181181141845040899006 12e-29
0.442380527886103165285271882501327525481283300279 94e-15
0.480843783896647982313265455230335254008016480655 79e-14
0.758288187019286238986333810690965567975835154399 22e-15
6.341174663247368263595514755447753852512733699549 8
0.126394889912650352952817821096259565163656471533 40e-14
0.199324094706330765254293269881548266422610341239 52e-15
145.461107240123910574049648787998984418028464984 55
.355839159471540020866871890549269076975980242043 28
```

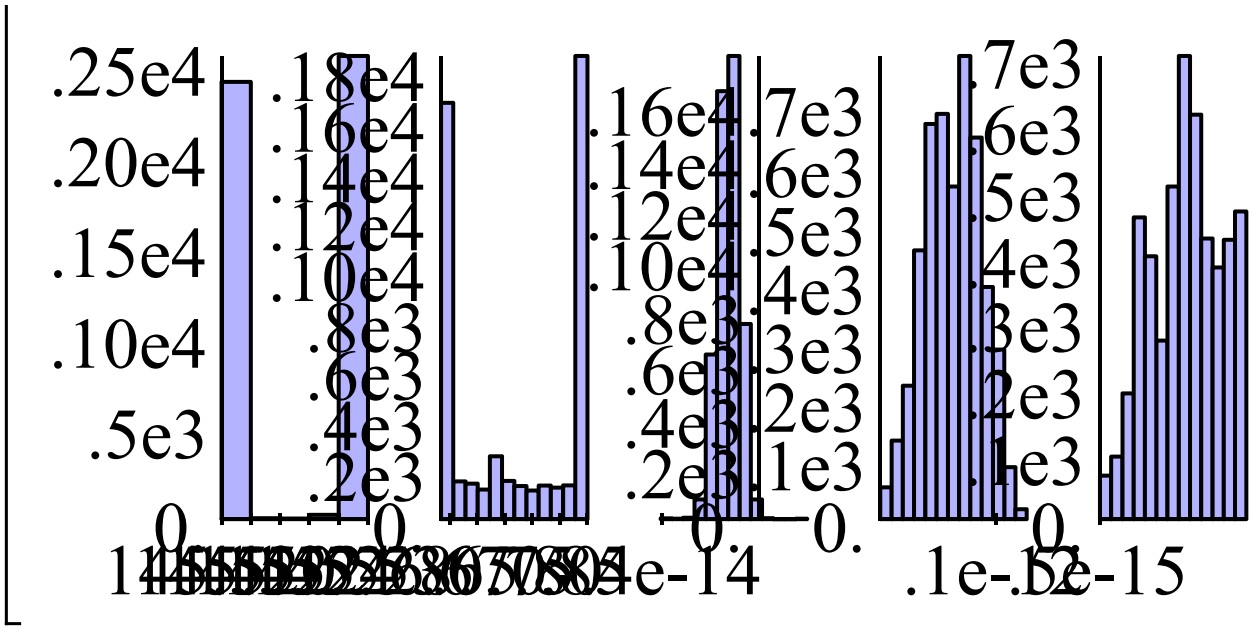
```
.725554225095780739692555665058661340304253891041 01
.11737243819795930910354954779762911785838165537722
0.16365430358483175864891852842350618639082388762678e-14
0.44238052788610316528527188250132752548128330027992e-15
-0.62938106948896663594192603763500379209200627927346e-15
0.51501892166450371686851147425320028923322948786637e-15
0.41139467450241511799801236419953061144251561212787e-13
0.84899739047132937777015005871941036453635918333789e-15
148.4362792
148.1751625
147.9139609
147.6526292
147.3912083
147.1296583
146.8679792
146.6061625
146.3441792
146.0819913
145.819675
145.5571542
145.2944333
145.0314792
144.7683208
144.5049
144.2412417
143.9773167
143.7131083
143.4486208
143.18385
142.9187957
142.6533458
-0.12457955999828941797106405624339989923841018435 2e-15
-0.88892146790571907235832117483472789391696096037 5e-15
0.46683189734051659987030237053348899582539652269e-16
0.38220596489668187777304560436392937908868909489 6e-15
0.41767259926791367973513495187195281625658393056 3e-15
0.85305798054307445610452419672851757443491968821 9e-15
0.68836210872216420688121333893362365362369636786 4e-15
0.32357963406609227584879687211595115629068118744 5e-15
0.55869230452384348297483248042040866732590347878 0e-15
-0.63237963852227524728456293424339434147362543 69e-17
0.82857929017638862876342131759149111135771939887 5e-15
```

```
0.16633536683092895163226381406507198078546922368 35e-14
0.12980021072901856616580029843098177400928674643 39e-14
0.32503774605676979973866170816939189446726835978e-16
0.66876859368671702406006421834571807525861810735e-16
-0.19891520546096748000712448854007099939004815695 7e-15
-0.64856748294610291996291606348652119706579976139e-16
-0.13096614705948581138726443583147049553446273339 6e-15
0.10274608758054549607713679284664536856297563476 7e-15
0.23628266196408243295344662996765730310204112397 7e-15
0.56964093269063314558179411944220724106680694780 9e-15
0.10282077388398373616885207288508765545651468786 1e-15
0.33575163192624449155473140038747966234607083022 8e-15
```

```
> 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=["Wavelength (microns)","Flux (W m^-2 um^-1)"], labelfont=[Arial,11,
  bold], font=[Arial,11,bold], labeldirections=[horizontal,vertical]);
```

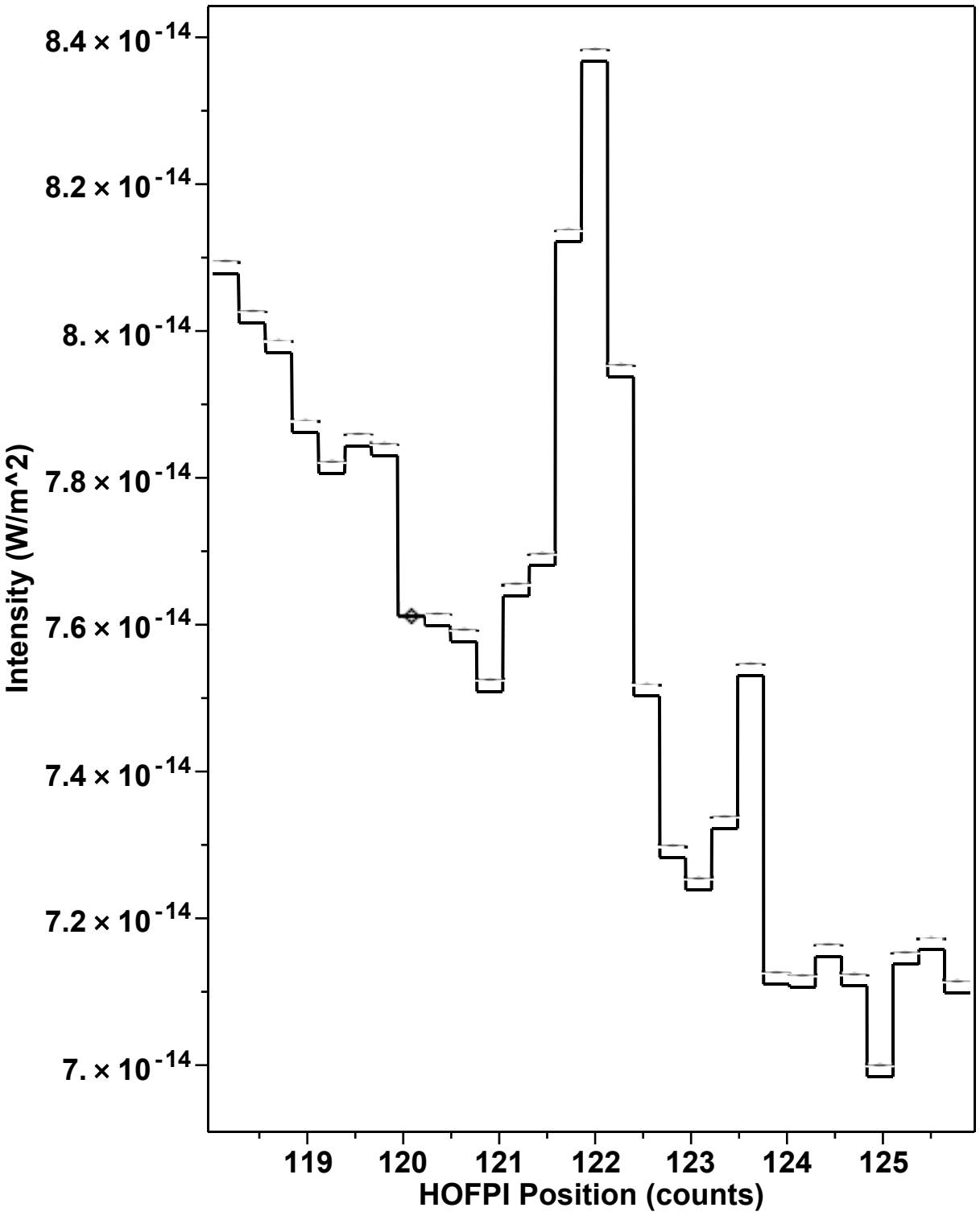



```
> display(array([Cpdf,Fpdf,Spdf,Opdf,Hpdf]));
```

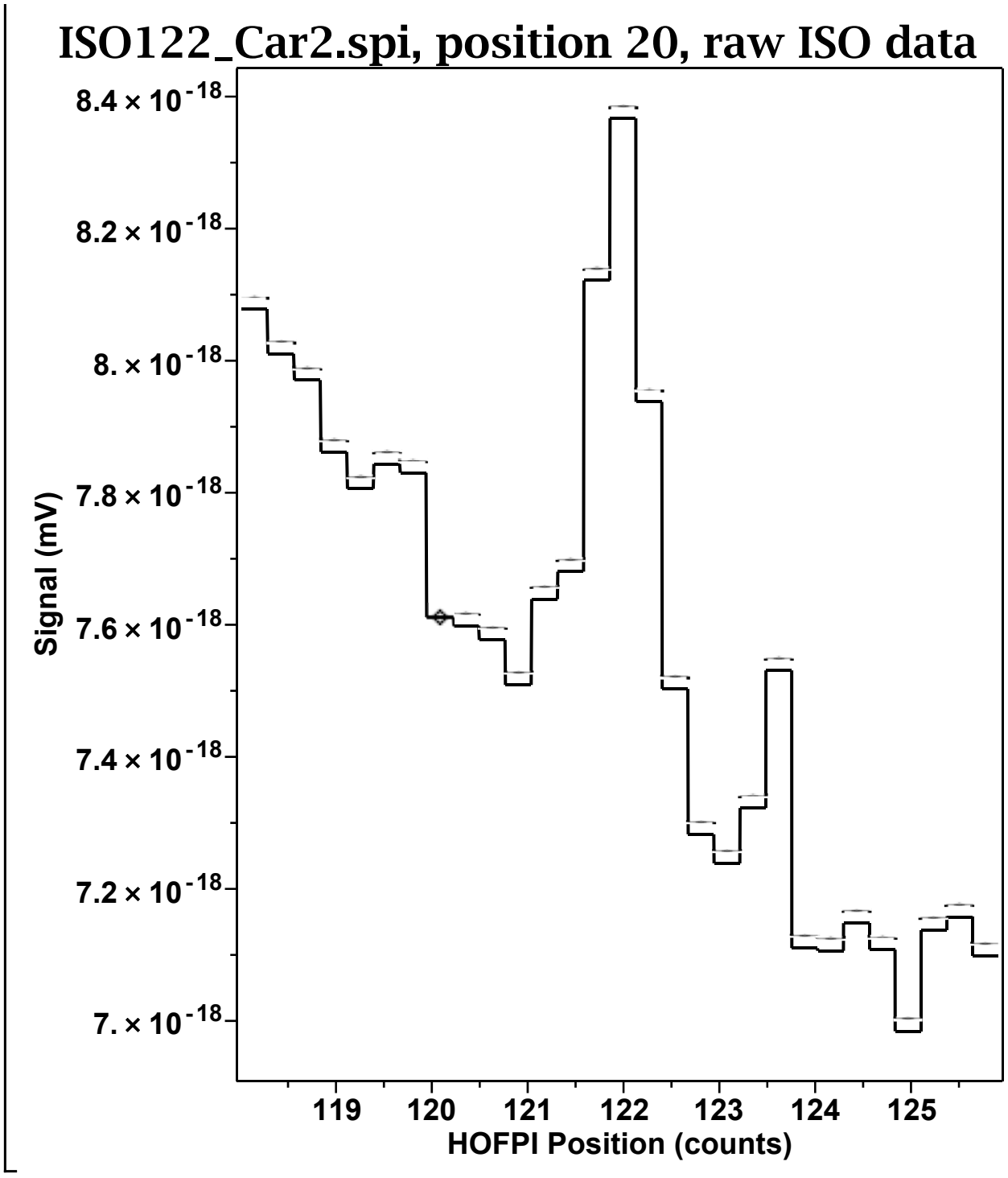



```
> display({graph2a,op(graph2b)},view=[XScaled[NumBins]
  -BinLengthScaled/2..XScaled[1]+BinLengthScaled/2,(min(op(convert
  (YScaled,list)))-0.05*(max(op(convert(YScaled,list)))-min(op
  (convert(YScaled,list))))))..(max(op(convert(YScaled,list)))+0.05*
  (max(op(convert(YScaled,list)))-min(op(convert(YScaled,list)))))
  ],axes=boxed,title=title2, titlefont=[Times,16,bold], labels=["HOFPI
  Position (counts)","Intensity (W/m^2)"], labelfont=[Arial,11,bold], font=
  [Arial,11,bold], labeldirections=[horizontal,vertical]);
```

ISO122_Car2.spi, position 20, scaled ISO data



```
> display({graph1a,op(graph1b)},view=[X[NumBins]-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]);
```


ISO122_Car2.spi, position 20, raw ISO data
8.4 × 10-18
8.2 × 10-18
8. × 10-18
7.8 × 10-18
7.6 × 10-18
7.4 × 10-18
7.2 × 10-18
7. × 10-18
Signal (mV)
119
120
121
122
123
124
125
HOFPI Position (counts)