Guide to Using R code

Inputs and description of the code:

We have written the code for this modeling method in the statistical programming software "R" (R Core Team; 2014). R is a user-friendly, open-source software package used for high-level statistical analyses and free for download on any computer platform. We have attached the input code as well as a "how-to" file with explicit directions. Below are step-by-step directions.

First download the R software package (http://www.r-project.org/) and install for your platform. A user-friendly software package called RStudio is also available but not necessary for use of our R script.

Start

Data should be saved in a .csv file in the EXACT following format (see the example data set in Table S1 as an example). The data should have a header row and the following columns, in this EXACT order: **Spot name, $^{207}Pb/^{235}U$ ratio, 1 sigma 7/35, $^{206}Pb/^{238}U$ ratio, 1 sigma 6/38, rho, $^{207}Pb/^{206}Pb$ Age** (in Myr). If any of these are incorrectly formatted the file *will not* read into the program properly.

Install Libraries:

Once R is opened, click on *"**Packages & Data**"* on the top menus and select "***Package Installer***". In the search bar type the following: ***RColorBrewer*** and click ***Get List***. You will be asked to select a CRAN mirror. Select whichever you prefer and then select the ***RColorBrewer*** package in the menu. Click ***Install Selected***.

Repeat this procedure for the packages ***cardidates, data.table*** and ***reshape2***.

Open the **.R file** in the supplementary information, or copy and paste the code from the **R code.docx** file into the R console. This code may be run line by line or all as one.

Define input parameters:

Input the starting parameters.

1. **Data Set Name:** This is changed in the ***DataTitle*** function (LINE 7). Within the quotations replace *"***YOURDATA***"* with *"**yourfilename***". Any characters will be accepted.
2. **File location**: In the ***read.csv()*** function (LINE 8) enter the location of the file to be modeled (note that R does not read single backslashes properly therefore only use "/" or "\\" to separate folders). Replace "**YOUR/FILE/LOCATION/IN/CSV/FORMAT**" with the location of the file to be modeled. Use the "\\" if you are running the code on a Windows platform and "/" on a Mac platform to separate directories.
3. **Working Directory:** In the ***setwd()*** function (LINE 9) put your preferred directory in place of "**YOUR/WORKING/DIRECTORY**".

One other parameter that may be changed is the node spacing (see Figure 2 for a graphical depiction of this). We find that 5–10 myr spacing is suitable for finding peaks, however, some may prefer a smaller or larger grid spacing depending upon the data set. This is changed in the **Tstep** function (Line 12). Currently this is set at 5 * 10^6 (5 myr) but this may be changed to any number.

Stacked probability plots:

One other output parameter we have defined is ***maximus,*** which is the maximum likelihood that will be plotted on the normalized upper and lower intercept plots. This is to be used for making stacked probability plots as many workers currently use for concordant probability density functions. This value will be the pre-defined height of each plot. Therefore, you should have some idea of what the maximum normalized likelihoods will be throughout the suite of sample to be compared and ***maximus*** should be greater than the maximum normalized likelihood.

Run:

Once these steps are complete, highlight the entire sequence of code and select **Edit -> Execute**. In our experience, an average-sized data set with ~150 analyses will run in under ten minutes on most computers using a 2 myr node spacing. This will obviously change with the number of analyses and the node spacing.

For example, using a MacBook Pro (2.9GHz processor, 16 GB RAM with 1867 MHz DDR3) the Sydgletcher data set (Morris *et al.,* 2015) containing 63 data points runs in <3 minutes.

Outputs:

The automatic outputs are as follows:

1. *YourTitle.csv*: a runfile containing relevant information regarding the modeling run including runtime and number of data points
2. *YourTitle results.csv:* The raw output data file containing upper intercept, lower intercept, summed probability density, and normalized likelihood. This may be used for plotting using other resources if necessary.
3. *YourTitle upper intercept.*csv: The file containing upper intercept ages and binned likelihood
4. *YourTitle lower intercept.*csv: The file containing lower intercept ages and binned likelihood
5. *YourTitle Peak Location and Width.csv*: The file containing all the significant local maxima (i.e., maxima that have likelihoods greater than 1/3 the maximum likelihood) in the two-dimensional likelihood map discussed below. Included with each maximum is the position of the upper intercept, positions of the lower intercept, and the width of the Gaussian fit in each direction. For both directions the number of points used in the Gaussian fit are used, as well as the likelihood measured at the maximum. Supplementary Figure S1 may help visualize this process.
6. *YourTitle 2D Histogram.pdf*: The two-dimensional likelihood map produced during the analysis

7. *YourTitle XYIntercepts.pdf*: This file contains the plot of normalized upper and lower intercept likelihoods
8. *YourTitle XYInterceptsnorm.pdf*: This file contains the plot of normalized upper and lower intercept likelihoods but with the maximum defined by the ***maximus*** input parameter. These plots can be used for creating stacked probability plots as workers currently do with probability density functions.
9. *YourTitle compare.pdf*: A file containing the normalized binned upper and lower intercept likelihoods with the maximum of the y-axis predefined in the code (see above) as well as the 2D heat map. To be used for stacked comparisons.
10. *YourTitle plate.pdf*: This is the final combined output containing several of the above plots. This is identical to Figures 3–5 in the paper.
11. *YourTitle Concordia.pdf*: A Concordia plot of the input data set.

Bootstrapping method for determining reproducibility:

True uncertainty propagation in a modelling method such as we proposed here is a complex and intricate issue. Therefore we have attempted to address this issue by performing a 'pick and replace' bootstrapping procedure to the Sydgletcher dataset, giving us an estimate of the reproducibility of the modelling procedure with variable data inputs. This entailed using the reported Sygletcher dataset of Morris et al. (2015) as the true dataset, and randomly sampling n = 63 analyses from this dataset. The 'pick and replace' selects each of the n = 63 selections at random from the Sydgletcher dataset, no matter which analysis was picked prior. In this way, we generate a random selection of 63 analyses from the Sydgletcher population (assumed to represent the true population), some of which will be duplicates. We performed this random sampling ~6000 times to produce ~6000 datasets randomly generated from the original Sydgletcher dataset. We then run our modeling procedure on each of the 6000 randomly generated datasets and record peak locations. This provides us with an effective reproducibility measure, in that the standard deviation of a population (n = 6000) about any one major peak gives us an estimate of the affect of sampling bias on the model outputs. Unfortunately, this bootstrapping method is very computationally expensive (6000 runs took over 4 days on a moderately fast desktop computer) and not likely to be widely used by the detrital zircon community. Nevertheless, we have included the code for bootstrapping as well as a description of the method in the supplementary materials so that interested parties may experiment with or improve the R code efficiency (see below for code). The results of our bootstrapped modelling procedure are documented below using two lower intercept peaks.

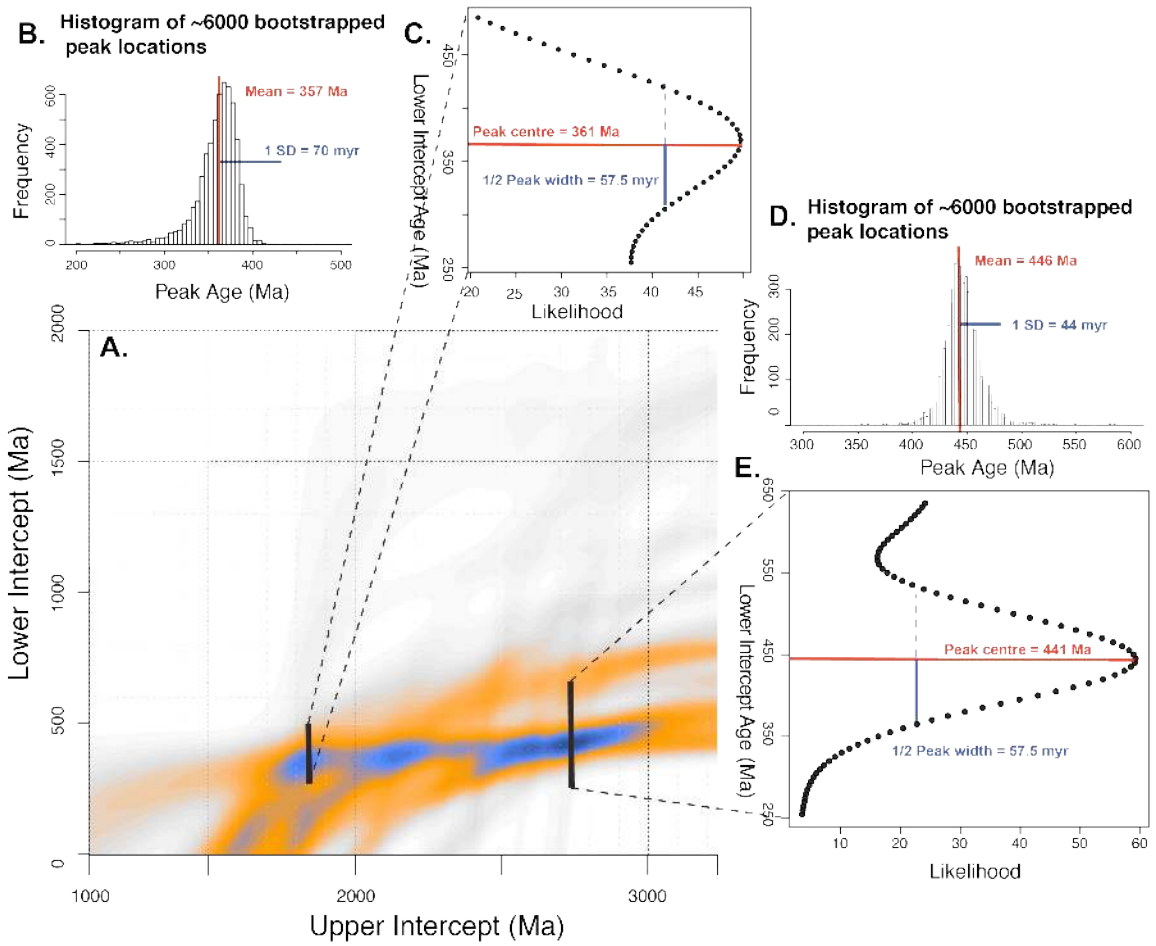

Figure S1: Results of bootstrapped resampling of the Sydgletcher datset. A. Results for the Sydgletcher dataset, expanded to show the region of high likelihood in more detail. B. A histogram of calculated peak centers for each bootstrapped dataset (n = ~6000) showing the reproducibility of the peak location given variable input data. C. A likelihood cross-section at a set upper intercept age across the peak of interest from the modelling output using the actual Sydgletcher dataset. This plot shows the likelihood at various lower intercept ages. Also shown are the Gaussian fit parameters of peak center and peak width at the fit. D. Histogram of peak locations of each of the n = ~6000 bootstrapped datasets. The standard deviation of the population of peak centers is around the same amount as the peak width measure defined by the Gaussian fit in (E). E. A plot showing how likelihood varies with lower intercept age across the highest peak in the dataset. Also shown are the Gaussian fit parameters of peak center and peak width.

R code for bootstrapping procedure:

Copy and paste the blue text into R in order to run the bootstrapping procedure.

```
rm(list = ls())

# Data should have format as follows
# sample name, ratio 7/5, 1sigma75, ratio 6/8, 1sigma 68, rho, age76 in Ma

# download data file and view and define Npoints for future calculations
Data.raw   <- read.csv("Your File")
DataTitle  <- "Your Title"
```

```
setwd("Your Location")

## Create the index and info for the bootstrapping loop
B             <- 5                          # number of times to run it through
N             <- length(Data.raw[, 1])            # number of data to draw (all of them)
P             <- 30                          # number of potential peaks
means         <- rep("mean", B * P)
stor.peaks.up   <- data.frame(matrix(NA, B, P))       # store upper intercept peaks
stor.peaks.dn   <- data.frame(matrix(NA, B, P))       # store lower intercept peaks
colnames(stor.peaks.dn)    <- rep("mean", P)
colnames(stor.peaks.up)    <- rep("mean", P)


# set input variables, these all are used for the main 'grid' not the dataset.
Tstep        = 10*10^6
# define the maximum of the y-axis
maximus      = 75
# number of data points in each block
Number       = 25

# activate necessary libraries
library(RColorBrewer)
library(reshape2)
library(cardidates)
library(data.table)
library(raster)

# definitions that should be outside the loop
deltaT              = 100*10^6
concTstep           = 100*10^3
f                 = 2
Tmin                = 0.0;
Tmax                = 4.5*10^9;
concNlines          <- Tmax/concTstep + 1;

Lambda238           <- 1.55125*10^(-10)
Lambda235           <- 9.8485*10^(-10)

# extra stuff outside the loop
Npoints             <- Number

# start the clock
ptm <- proc.time()

# Creating files and equations outside the loop
a                   <- as.vector( seq( from = 0, to = Tmax - Tstep, by = Tstep ))
b                   <- as.vector( seq( from = Tmin + Tstep, to = Tmax, by = Tstep))
DiscGrid                <- expand.grid( a, b )
DiscGrid2               <- subset( DiscGrid, Var1 < Var2)
DiscGrid3               <- DiscGrid2[ with( DiscGrid2, order( Var1 )), ]
DiscGridTable           <- DiscGrid3
colnames(DiscGridTable)    <- c( "Lower Intercept", "Upper Intercept" )

aff          <- function(x1, y1, x2, y2)  {(y2 - y1)/(x2 - x1)
}
bff          <- function(x1, y1, x2, y2)  {y2 - x2*(y2 - y1)/(x2 - x1)
```

```
}
afff        <- function(t1, t2) {aff(exp(Lambda235 * t1) - 1, exp(Lambda238 * t1) - 1,
                          exp(Lambda235 * t2) - 1, exp(Lambda238 * t2) - 1)
}
bfff        <- function(t1, t2) {bff(exp(Lambda235 * t1) - 1, exp(Lambda238 * t1) - 1,
                          exp(Lambda235 * t2) - 1, exp(Lambda238 * t2) - 1)
}

DiscGridTableA              <- mapply(afff, DiscGridTable["Lower Intercept"],
                          DiscGridTable["Upper Intercept"])
colnames(DiscGridTableA)    <- "Slope"
DiscGridTableB              <- mapply(bfff, DiscGridTable["Lower Intercept"],
                          DiscGridTable["Upper Intercept"])
colnames(DiscGridTableB)    <- "Yintercept"
DiscGridTableFinal          <- cbind(DiscGridTable[1:2], DiscGridTableA, DiscGridTableB)
DiscGridTableFinal          <- DiscGridTableFinal[c(3, 4, 1, 2)]
row.names(DiscGridTableFinal) <- seq_len(nrow(DiscGridTableFinal))
DiscGridTableFinal$ID       <- seq(1, nrow(DiscGridTableFinal), 1)
Discline                    <- nrow(DiscGrid)
Disclines                   <- Discline
Pro  <- function(a, b, Xi, sX, Yi, sY, rho, disc) { abs(disc) * (1 / (2 * pi * sX * sY)) *
   exp((-1 / 2) * ((((b + a * Xi - Yi) / (cos(atan((2 * rho * sX * sY) /
                                    (sX ^ 2) - (sY ^ 2)) / 2) + a * sin(atan((2 * rho * sX * sY) /
                                                                  (sX ^ 2) - (sY ^ 2)) / 2))) / sY) ^ 2 / (1 + (sX / sY
* ((a *

cos(atan((2 * rho * sX * sY) / (sX ^ 2) - (sY ^ 2)) / 2) -

sin(atan((2 * rho * sX * sY) / (sX ^ 2) - (sY ^ 2)) / 2)) /

(cos(atan((2 * rho * sX * sY) / (sX ^ 2) - (sY ^ 2)) / 2) + a *

sin(atan((2 * rho * sX * sY) / (sX ^ 2) - (sY ^ 2)) / 2)))) ^ 2)))
}
Prob   <- function(p1, p2) {
  p1 = as.list(p1); p2 = as.list(p2)
  Pro(p1$Slope, p1$Yintercept, p2$r75, p2$sigma75, p2$r68,
      p2$sigma68, p2$rho, p2$discordance)
}

BigFunction  <- function (x) {
  Npoint                <- dim(Data.new)
  Npoints               <- (Npoint[1])
  indexdisc      <- CJ(indexdisc1 = seq( nrow( DiscGridTableFinal )),
                       indexdisc2 = seq( nrow( x )))
  sumdisc        <- indexdisc[, `:=`(resultdisc = Prob( DiscGridTableFinal[indexdisc1, ],
                                        x[indexdisc2, ]),
                                     Group.1 = rep( seq( nrow( DiscGridTableFinal )),
                                                    each = nrow( x )))][,.(sumdisc = sum( resultdisc )),
                                                                        by = Group.1]
  sumdisc               <- as.data.frame( sumdisc )

  colnames(sumdisc)      <- c("ID", "Likelihood")
  Resultdisc             <- merge(DiscGridTableFinal, sumdisc, by = "ID", all.x = TRUE)
  row.names(Resultdisc)  <- seq_len(nrow(Resultdisc))
  rm(indexdisc, sumdisc)
```

```
  assign(paste("Resultdisc"), Resultdisc)
}


# Gaussian fit function
fitG =
  function(x,y,mu,sig,scale){

    f = function(p){
      d = p[3] * dnorm( x, mean = p[ 1 ], sd = p[ 2 ] )
      sum( ( d - y ) ^ 2)
    }
    optim( c( mu, sig, scale ), f )
  }

# try to fit the gaussian fit function to the data array
# x is data array, y is the lower intercept location,
# z is the upper intercept location
fitG.upper   <- function (x, y, z) {
  i <- 1
  repeat {
    i               <- i + 1
    z1              <- z - 100 + i
    z2              <- z + 100 - i
    if (z2 > nrow(aa)) {
      z2 <- nrow(aa)
    }
    newdata         <- data.frame( x[ z1:z2, y] )
    rows            <- c( seq( z1 * Tstep, z2 * Tstep, Tstep ))
    newdata         <- data.frame( cbind( rows, newdata ))
    n               <- nrow( newdata )
    colnames(newdata) <- c( "Intercept", "Likelihood" )
    newfit          <- fitG( newdata$Intercept, newdata$Likelihood, 3.0e9, 200e7, 1)
    pred.likelihood   <- data.frame( Intercept = newdata$Intercept,
                                     Predicted = newfit$par[3]* dnorm(newdata$Intercept,
                                                                      newfit$par[1], newfit$par[2]))
    deviates        <- pred.likelihood$Predicted - newdata$Likelihood
    deviates.2      <- deviates ^ 2
    sum.dev.2       <- sum(deviates.2) / x[z, y]
    if (sum.dev.2 < 0.05 & n > 11 |
        sum.dev.2 < 1 & n > 3 & n < 11 |
        i > 98)
      break
  }
  list(mean = newfit$par[1])
}
fitG.lower   <- function (x, y, z) {
  i <- 1
  repeat {
    i               <- i + 1
    y1              <- y - 100 + i
    y2              <- y + 100 - i
    if (y1 < 1) {
      y1 <- 1
    }
    newdata         <- data.frame( x[ z, y1:y2 ] )
```

```
   rows             <- c( seq( y1 * Tstep, y2 * Tstep, Tstep ))
   newdata          <- data.frame( cbind( rows, newdata ))
   n               <- nrow( newdata )
   colnames(newdata) <- c( "Intercept", "Likelihood" )
   newfit           <- fitG( newdata$Intercept, newdata$Likelihood, 1.0e9, 200e7, 1)
   pred.likelihood   <- data.frame( Intercept = newdata$Intercept,
                            Predicted = newfit$par[3]* dnorm(newdata$Intercept,
                                                  newfit$par[1], newfit$par[2]))
   deviates         <- pred.likelihood$Predicted - newdata$Likelihood
   deviates.2       <- deviates ^ 2
   sum.dev.2        <- sum(deviates.2) / x[z, y]
   if (sum.dev.2 < 0.05 & n > 11 |
      sum.dev.2 < 1 & n > 3 & n < 11 |
      i > 98)
     break
 }
 list(mean = newfit$par[1], width = round(n * Tstep / 1e6),
     points =  n, deviation = round(sum.dev.2, 6))
}



# analyze the main data set to find the dominate peaks, then use those as a guide
# within the for loop

Data.new             <- Data.raw
colnames(Data.new)    <- c("Spot", "r75", "sigma75", "r68", "sigma68", "rho", "age76")
Data.new["sigma75"]   <- median( Data.new[ , "sigma75"])
Data.new["sigma68"]   <- median( Data.new[ , "sigma68"])

datapoints           <- nrow(Data.new)

discordance          <- matrix( c( abs(1 - (Data.new$r68 / (exp( Lambda238 * Data.new$age76 *
                                          1000000) - 1)))))
Data.new             <- data.frame(cbind(Data.new$Spot, Data.new$r75, Data.new$sigma75,
                            Data.new$r68,
                            Data.new$sigma68, Data.new$rho, discordance))
colnames(Data.new)    <- c("Spot", "r75", "sigma75", "r68", "sigma68", "rho", "discordance")

nfiles               <- ceiling(nrow(Data.new)/Number)
grouping             <- as.data.frame(c(rep(1:(nfiles - 1), each = Number),
                            rep(nfiles, times = nrow(Data.new) -
                                (as.integer(Number*(nfiles - 1))))))
Data.new             <- cbind(Data.new, grouping)
colnames(Data.new)    <- c("Spot", "r75", "sigma75", "r68", "sigma68", "rho", "discordance", "GROUP")

data.split           <- split(Data.new, Data.new$GROUP)

bigdata              <- by(Data.new[, 1:7], Data.new$GROUP, BigFunction)
splitfun   <- function(x) {
  bigdata[[x]]$Likelihood
}
for (k in 1:nfiles) {
  assign(paste("res", k), as.data.frame(splitfun(k)))
}
```

```
likelis              <- do.call(cbind, lapply(paste("res", 1:nfiles, sep=" "), get))
totallikelihood      <- apply(likelis, 1, sum)
Resultdisc           <- cbind(bigdata$`1`[, 1:5], as.data.frame(totallikelihood))
colnames(Resultdisc)  <- c("ID", "Slope", "Yintercept", "Lower Intercept", "Upper Intercept",
                     "Likelihood")
normalized           <- Resultdisc [, "Likelihood"] / datapoints
Resultdisc           <- cbind(Resultdisc, normalized)
upperdisc            <- aggregate (Resultdisc$normalized,
                           by = list (Resultdisc [, "Upper Intercept"]), max)
colnames(upperdisc)   <- c("Upper Intercept", "Likelihood")

lowerdisc            <- aggregate (Resultdisc$normalized,
                           by = list(Resultdisc[, "Lower Intercept"]), max)
colnames(lowerdisc)   <- c("Lower Intercept", "Likelihood")


### Uncertainty analysis



# Fit the Gaussian curves to the peaks identified in the 'lowerpeak' file
aa          <- acast(Resultdisc, `Upper Intercept`~`Lower Intercept`, value.var = "normalized",
                fun.aggregate = mean)

# filter by peak height, only using data that are > 1/3 the max value
aaa         <- ifelse( aa < max(aa / 3, na.rm = TRUE), NA, aa)
colnames(aa) <- c(seq(1, ncol(aa), 1))
rownames(aa) <- c(seq(1, nrow(aa), 1))

## Convert it to a raster object
r        <- raster(aaa)
extent(r) <- extent(c( 0, ncol(aaa), 0, ncol(aaa) ) + 0.5)

## Find the maximum value within the 9-cell neighborhood of each cell and put it in the
# group of 9 cells
f         <- function( X ) {max (X, na.rm = FALSE)}
localmax   <- focal( r, w = matrix( 1, 3, 3 ), fun = f, pad = TRUE, padValue = NA)

## Does each cell have the maximum value in its neighborhood?
r2        <- r == localmax

## Get x-y coordinates of those cells that are local maxima
maxXY           <- data.frame(xyFromCell( r2, Which(r2==1, cells=TRUE)))
colnames(maxXY)  <- c("A", "B")
heights         <- rep(0, nrow(maxXY))
for (m in 1:nrow(maxXY))  {
  heights[m]  <- aa[nrow(aa) - maxXY$B[m], maxXY$A[m]]
}
maxXY           <- data.table(cbind(maxXY, heights))
maxXY           <- setorder(maxXY, -`heights`)


uncert  <- data.frame(       Upeak = rep(0, nrow(maxXY)),
                        Uwidth = rep(0, nrow(maxXY)),
```

```
                        Upoints = rep(0, nrow(maxXY)),
                        Udeviation = rep(0, nrow(maxXY)),
                        Lpeak = rep(0, nrow(maxXY)),
                        Lwidth = rep(0, nrow(maxXY)),
                        Upoints = rep(0, nrow(maxXY)),
                        Udeviation = rep(0, nrow(maxXY)),
                        `normalized likelihood` = rep(0, nrow(maxXY)))

for (i in 1:nrow(maxXY)) {
  uncert[i, 5:8] <- fitG.lower( aa, maxXY$A[i], nrow(aa) - maxXY$B[i])
  uncert$normalized.likelihood[i]  <- aa[nrow(aa) - maxXY$B[i], maxXY$A[i]]
}
uncert$Lpeak            <- round(uncert$Lpeak / 1e6)

for (i in 1:nrow(maxXY)) {
  uncert[i, 1:4] <- fitG.upper( aa, maxXY$A[i], nrow(aa) - maxXY$B[i])
}
uncert$Upeak            <- round(uncert$Upeak / 1e6)


# remove stuff
# rm(aa, r2, r, f, localmax, aaa, lowerdisc, upperdisc, Resultdisc, normalized)




#### Start the bootstrapping loop



for ( i in 1:B)  {
  indx    <- sample(1:N, N, replace = TRUE)

# Start dealing with the data
Data.new    <- Data.raw[ indx, ]
colnames(Data.new)    <- c("Spot", "r75", "sigma75", "r68", "sigma68", "rho", "age76")
Data.new["sigma75"]   <- median( Data.new[ , "sigma75"])
Data.new["sigma68"]   <- median( Data.new[ , "sigma68"])

datapoints          <- nrow(Data.new)

discordance          <- matrix( c( abs(1 - (Data.new$r68 / (exp( Lambda238 * Data.new$age76 *
                                                1000000) - 1)))))
Data.new             <- data.frame(cbind(Data.new$Spot, Data.new$r75, Data.new$sigma75,
                                Data.new$r68,
                            Data.new$sigma68, Data.new$rho, discordance))
colnames(Data.new)    <- c("Spot", "r75", "sigma75", "r68", "sigma68", "rho", "discordance")

nfiles              <- ceiling(nrow(Data.new)/Number)
grouping             <- as.data.frame(c(rep(1:(nfiles - 1), each = Number),
                       rep(nfiles, times = nrow(Data.new) -
                           (as.integer(Number*(nfiles - 1))))))
Data.new             <- cbind(Data.new, grouping)
colnames(Data.new)    <- c("Spot", "r75", "sigma75", "r68", "sigma68", "rho", "discordance", "GROUP")

data.split          <- split(Data.new, Data.new$GROUP)
```

```
bigdata              <- by(Data.new[, 1:7], Data.new$GROUP, BigFunction)
splitfun   <- function(x) {
  bigdata[[x]]$Likelihood
}
for (k in 1:nfiles) {
  assign(paste("res", k), as.data.frame(splitfun(k)))
}


likelis              <- do.call(cbind, lapply(paste("res", 1:nfiles, sep=" "), get))
totallikelihood       <- apply(likelis, 1, sum)
Resultdisc           <- cbind(bigdata$`1`[, 1:5], as.data.frame(totallikelihood))
colnames(Resultdisc)  <- c("ID", "Slope", "Yintercept", "Lower Intercept", "Upper Intercept",
                      "Likelihood")
normalized            <- Resultdisc [, "Likelihood"] / datapoints
Resultdisc           <- cbind(Resultdisc, normalized)
upperdisc            <- aggregate (Resultdisc$normalized,
                            by = list (Resultdisc [, "Upper Intercept"]), max)
colnames(upperdisc)   <- c("Upper Intercept", "Likelihood")

lowerdisc            <- aggregate (Resultdisc$normalized,
                            by = list(Resultdisc[, "Lower Intercept"]), max)
colnames(lowerdisc)   <- c("Lower Intercept", "Likelihood")


### Uncertainty analysis using x,y coordinates from the real data set analysis
### This allows for the identification of the same peak everytime, although some will be
### way off.  Oh well.

# Fit the Gaussian curves to the peaks identified in the 'lowerpeak' file
aa          <- acast(Resultdisc, `Upper Intercept`~`Lower Intercept`, value.var = "normalized",
                fun.aggregate = mean)

for (j in 1:nrow(maxXY)) {
  stor.peaks.dn[i, j] <- fitG.lower( aa, maxXY$A[j], nrow(aa) - maxXY$B[j])
}
for (j in 1:nrow(maxXY)) {
  stor.peaks.up[i, j] <- fitG.upper( aa, maxXY$A[j], nrow(aa) - maxXY$B[j])
}


####  End the loop
}


# Gaussian fit of each peaks location

# t.test(stor.peaks.dn$mean.2, alternative = c("two.sided"))

setwd("C:/Users/jwaldronlab/Desktop")

write.table(stor.peaks.dn, file = paste(DataTitle, "Peak Location Lower.csv"),
          row.names = FALSE, quote = FALSE, sep = ",")
write.table(stor.peaks.up, file = paste(DataTitle, "Peak Location Upper.csv"),
          row.names = FALSE, quote = FALSE, sep = ",")
```

```
# Stop the clock
runtime = proc.time() - ptm

runfile  <- matrix( c( "Data set Title", DataTitle, "Bootstrapping number", B,
                   "Node Spacing (myr)", Tstep, "Number of gridlines", Discline,
                   "Run time (sec)", runtime[3]), 5, 2, byrow = TRUE)
colnames(runfile)    <- c( "Information", "This run")

write.table(runfile, file = paste(DataTitle, ".csv"), row.names = FALSE,
          col.names = FALSE, quote = FALSE, sep = ",")
## END!
```

End of the code for bootstrapping procedure.