Introduction to R for CCP4 users

James Foadi
Membrane Protein Laboratory – Imperial College London and Diamond Light Source Ltd
Harwell Science and Innovation Campus, Didcot, Oxfordshire OX11 0DE, UK

1. Introduction.

Protein crystallographers are used to assess their data through the analysis of specific statistics and graphics. It is, for instance, the norm to report R_{meas}, completeness, I over sigI and multiplicity as overall numbers and in plots as functions of resolution, when presenting datasets quality in research papers. In CCP4 access to such statistics is made available through log files, log graphs or annotated logs in a web browser. This is certainly more than enough for the average requirement of users. There might be several situations, though, where the analysis provided by individual programs is not sufficient. In such situations users prefer to scrutinize data by themselves rather than to rely on the available statistical analysis. While accessing data is relatively straightforward in the CCP4 suite, flexible programs that allow data analysis are mostly missing. Such analysis should be carried out with the help of *ad hoc* software which would take a considerable amount of time to code, not to mention the programming skills required. This is certainly a task beyond the possibilities and interest of the average CCP4 user.

An obvious way out is the utilization of generic and flexible platforms for statistical data analysis. Many packages have been developed for the purpose, and some of them have become well established in the communities of professional statisticians and people making heavy use of statistics [1, 2, 3, 4, 5]. Among these many packages, the *R software* [4] stands out as one of the most popular and flexible environment for statistical calculations. R has been spreading beyond the international community of statisticians in many academic disciplines. The main reason for this is the open-source nature of the platform, joined to an informative and collaborative spirit present within the community of R enthusiasts. Furthermore, the amount of documentation and freely-available training material for this software is vast and increasing [4]. Furthermore, there are hundreds of add-on packages for a large variety of tasks and disciplines available for download, free of charge.

In crystallography R appears to be timidly used as a side program to simulate or test data, and to carry out all sorts of statistical analysis. It is probably also employed to generate publication-quality graphs. For instance a quick search of the IUCr-journals website returns 9 hits for papers where R has been explicitly referenced [6]. One of the reasons why R is still not used in crystallography as abundantly as in other fields is possibly connected with the specific nature of crystallographic file formats. R can read numerous formats, but R packages-that deal with crystallography have either been not completed [7, 8] or are entirely absent from the pool of available R packages.

In this short article it will be shown how simple and involved analysis can be carried out without much effort thanks to the powerful and widely applicable R built-in functions. The main purpose of this article is to encourage crystallographers to use R for data analysis and graphical data representation.

2. R in a (very small) nutshell

R can be installed on all common platforms including Windows, Mac OS X and Linux. The examples below were generated using R version 2.12.1 installed on Linux Ubuntu 12.04LTS.
R can be started typing “R” in the terminal. A welcoming message, similar to the one shown here,

is displayed:

```
R version 2.12.1 (2010-12-16)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]
```

From this message we can learn how to use the help system. There are two options, in-line help and help via a browser. Data created during a session are stored in the computer active memory, in an area known as *workspace*. Anything created in the workspace is referred to as an *object*. Commands are more correctly known as *functions*. Function “ls” can be used to view what objects are available in the workspace. For instance since nothing was created in the current R session, the following output is obtained:

```
> ls()
character(0)
```

“ls” is followed by brackets, “()”; all functions need brackets. They can include numbers, characters or other R objects, but they must always be present, even if they do not require any argument. Function “ls” returns a vector whose elements are characters indicating the name of the objects in the workspace; in the above case there are no objects and “ls” returns a message saying that the vector character has size zero. Once objects start filling the workspace, the “ls” output changes. For instance let us create 10 gaussian deviates and the alphabet lowercase letters up to letter “h”:

```
> obj1 <- rnorm(10,mean=1,sd=2)
> length(obj1)
[1] 10
> obj2 <- letters[1:8]
> length(obj2)
[1] 8
> length(letters)
[1] 26
```

“rnorm” is a function to generate gaussian random deviates. R has several built-in random generators for a variety of probability distributions. In the above example the 10 numbers created by this function are stored in the object “obj1” using the *assign* operator “<-” (a “less” symbol followed by a “minus” symbol). The “obj2” object is a character vector containing the first 8 lowercase alphabet letters; these are extracted by the global constant “letter”, which is a character vector containing the 26 lowercase alphabet letters. This time the “ls” function will return a

character vector with 2 names, as the current workspace has been filled with two objects:

```
> ls()
[1] "obj1" "obj2"
```

Many built-in functions exist to load data inside the workspace. One of the most used data readers is the “read.table” function. This is suitable to read equal-length columns of numbers or characters. In the following example we read in the content of an mtz file, previously converted into an ascii file with the CCP4 program “mtz2various”:

```
> mtzdata <- read.table("insulin.dat")
> str(mtzdata)
'data.frame': 7007 obs. of  8 variables:
 $ V1: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V2: int  1 1 1 1 1 1 1 1 1 1 ...
 $ V3: int  3 5 7 9 11 13 15 17 19 21 ...
 $ V4: num  873 4896 1384 5135 438 ...
 $ V5: num  12.8 44.4 11.8 42.7 4.8 ...
 $ V6: num  295 700 372 717 209 ...
 $ V7: num  2.17 3.17 1.58 2.98 1.15 ...
 $ V8: int  2 19 17 8 9 11 18 4 11 13 …
```

Data are loaded in the object named “mtzdata”. The content of this object can be explored and summarised with the function “str”. The object is of the type known as *data.frame*. This is the typical container for statistical datasets. A data.frame is very similar to a matrix with some annotations included. In other words, a data.frame is a matrix with mixed data types. Each column in the “mtzdata” data.frame has a name. All column names can be easily visualised:

```
> colnames(mtzdata)
[1] "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8"
```

These are generic names, as no headers were included in the original file. We know the correct names from the CCP4 original mtz file. Column labels can, thus, be assigned as follows:

```
> colnames(mtzdata) <- c("H","K","L","IMEAN","SIGIMEAN","F","SIGF"," FreeR_flag" )
```

The first 5 rows of the data.frame bears some similarity with part of the mtz content, as displayed by the program “mtzdump”:

```
> mtzdata[1:5,]
   H  K  L     IMEAN    SIGIMEAN         F     SIGF    FreeR_flag
1  0  1  3  873.04822   12.79639  295.43994  2.16591  2
2  0  1  5  4895.71387  44.41380  699.62976  3.17424  19
3  0  1  7  1383.53491  11.76934  371.94446  1.58220  17
4  0  1  9  5135.15283  42.72186  716.54718  2.98121  8
5  0  1 11  437.75629   4.80070   209.21368  1.14740  9
>
```

An important feature of R is the ability to produce simple, intermediate and complicated graphics of publication quality. Let us create a simple plot using simulated data. To simulate the behaviour of a response variable “y”, function of an explanatory variable “x”, we first generate a regular grid of 50 points, say between 0 and 10, using the function “seq”:

```
> x <- seq(0,10,length=50)
> str(x)
 num [1:50] 0 0.204 0.408 0.612 0.816 ...
```

Then a linear relation is imposed on y with added gaussian noise:

```
> y <- 2*x+1+rnorm(50, mean=0, sd=1)
```

The above line, among other things, shows that operations in R are vectorised. “x” and “y” are vectors containing 50 numbers each; when “2*x” is typed all 50 numbers are multiplied by two. Furthermore, adding 1 in this context means adding 1 to all 50 numbers. The expression is completed by adding 50 gaussian random deviates with mean 0 and standard deviation 1 to the 50 modified values. The linear relation can be visualised with a plot using the “plot” function (Figure 1):

```
> plot(x, y)
```

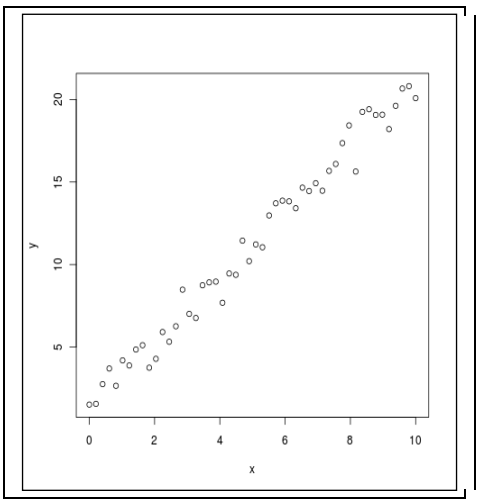


Figure 1

The “plot” command with no parameters produces the basic plot shown in Figure 1. We can enrich this plot. For instance, we can add a title using the keyword “main”, replace the open circles with full circles using the keyword “pch” and add the straight line interpolating simulated points with the “curve” function (see Figure 2):

```
> plot(x, y, main="Linear Regression", pch=16)
> curve(2*x+1, col="red", add=TRUE)
```

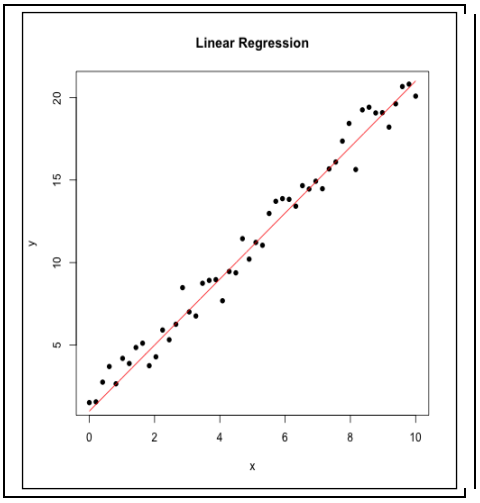


Figure 2

The “ADD=TRUE” keyword included in “curve” means that the straight line is to be added to the existing active plot. More features can be added and figures can be precisely tailored to fit any reasonable requirements. Indeed the R graphics capability is one of the main reasons why this platform has become very popular.

To conclude this very sketchy introduction to R let us illustrate the mechanism through which statistical modelling is performed. Statistical modelling implies the formulation of simple or complex relationships between explanatory and response variables that should explain data variability with a given degree of precision. The process consists of the creation of a model, the fitting of model to data and, finally, the analysis of the results. These different steps can be illustrated by a simple linear model with just one explanatory variable, x, and one response variable, y. The data previously introduced in the plotting example can also be used to illustrate linear regression. Many models can be speculated if the relationship between x and y is not known, but it makes sense to start with simple analytic forms with very few parameters and continue with an increase in analytic complexity and number of parameters until a satisfactory result is met. The definition of a model follows a “regression grammar”, where the symbol “~” is a substitute for regression, while symbols “+” and “-“ means inclusion or subtraction of terms in the regression model. Thus, for instance, the expression,

$$y \sim x$$

is a substitute for a regression where the model is a straight line with intercept, while the expression,

$$y \sim x - 1$$

is a substitute for a regression where the model is a straight line without intercept. The fitting itself is carried out by the function “lm”, in which the model is specified. The results can be saved into an R object of a class called class *lm*. Such object can, subsequently, be “queried” or used in a variety of ways for model assessment, for instance with function “summary”, for a quick report of the fitting:

```
> lm_obj <- lm(y ~ x)
> summary(lm_obj)

Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-1.79056 -0.40417  0.00207  0.55003  1.47165

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.39802    0.22875   6.112 1.69e-07 ***
x            1.96340    0.03942  49.808  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.8209 on 48 degrees of freedom
Multiple R-squared: 0.981,   Adjusted R-squared: 0.9806
F-statistic:  2481 on 1 and 48 DF,  p-value: < 2.2e-16
```

The “summary” output is quite informative. For example we can read that the r-squared goodness of fit (squared correlation coefficient) is 0.981, which indicates a very good fit. Among other things we also learn that the estimated slope of the model is 1.96340, while the intercept is estimated as 1.39802. The estimated straight line can be overlapped on the correct straight line using another R function, “abline” and using the model object as an input (see Figure 3):

```
> abline(lm_obj,col="green")
```

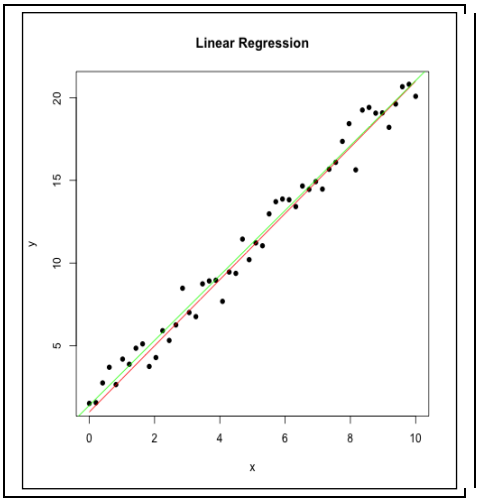


Figure 3

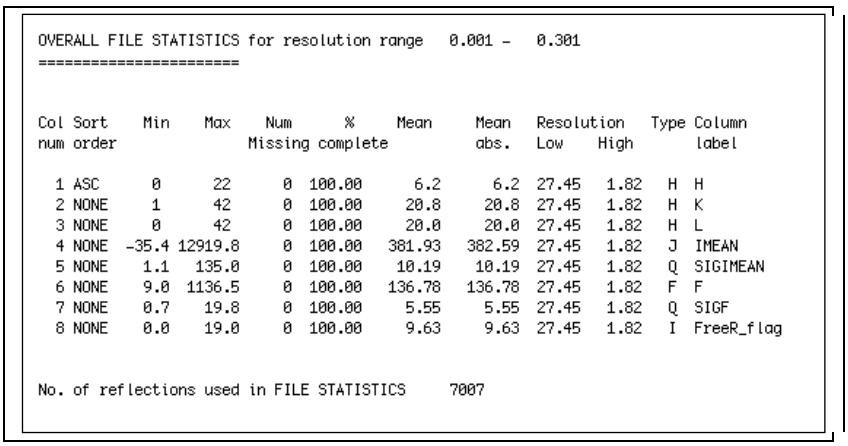
The green line in Figure 3 is the estimated line; it does not coincide with the true red line because of the random errors introduced during the simulation.

All data and typed commands created during an R session can be saved in a special file, which can be reloaded at the beginning of the next R-session. Each session file and the directory where it is stored can be associated with specific projects.

A lot of introductory material on R, including tutorials and books, can be found browsing the platform’s official web site [4].

3. Standard analysis of an mtz reflection file

In Section 2 the content of an mtz file translated in ascii form has been loaded in a data.frame called “mtzdata”. This R object will be now employed to carry out data analysis using built-in R functions. The first feeling for the data contained in the mtz file is normally acquired using the CCP4 program *mtzdump*. One section of the output is displayed in Figure 4.



```
OVERALL FILE STATISTICS for resolution range   0.001 -   0.301
=======================


Col Sort    Min    Max    Num      %     Mean     Mean   Resolution   Type Column
num order              Missing complete         abs.   Low    High       label

  1 ASC      0      22      0  100.00     6.2      6.2  27.45   1.82   H  H
  2 NONE     1      42      0  100.00    20.8     20.8  27.45   1.82   H  K
  3 NONE     0      42      0  100.00    20.0     20.0  27.45   1.82   H  L
  4 NONE  -35.4 12919.8     0  100.00   381.93   382.59  27.45   1.82   J  IMEAN
  5 NONE   1.1   135.0     0  100.00    10.19    10.19  27.45   1.82   Q  SIGIMEAN
  6 NONE   9.0  1136.5     0  100.00   136.78   136.78  27.45   1.82   F  F
  7 NONE   0.7    19.8     0  100.00     5.55     5.55  27.45   1.82   Q  SIGF
  8 NONE   0.0    19.0     0  100.00     9.63     9.63  27.45   1.82   I  FreeR_flag


No. of reflections used in FILE STATISTICS     7007
```

Figure 4

This is easily re-producible in R using the polymorphic function “summary” on the data.frame:

```
> summary(mtzdata)
H                K                L                IMEAN
Min.   : 0.000   Min.   : 1.00    Min.   : 0.00    Min.   :  -35.35
1st Qu.: 2.000   1st Qu.:14.00    1st Qu.:13.00    1st Qu.:   22.32
Median : 5.000   Median :21.00    Median :20.00    Median :   77.37
Mean   : 6.223   Mean   :20.79    Mean   :20.04    Mean   :  381.93
3rd Qu.:10.000   3rd Qu.:28.00    3rd Qu.:27.00    3rd Qu.:  304.54
Max.   :22.000   Max.   :42.00    Max.   :42.00    Max.   :12919.85

SIGIMEAN         F                SIGF             FreeR_flag
Min.   :  1.076  Min.   :   9.042 Min.   : 0.6535  Min.   : 0.000
1st Qu.:  6.276  1st Qu.:  44.042 1st Qu.: 2.6008  1st Qu.: 5.000
Median :  8.130  Median :  87.206  Median : 4.3532  Median :10.000
Mean   : 10.188  Mean   : 136.775  Mean   : 5.5494  Mean   : 9.628
3rd Qu.: 10.844  3rd Qu.: 174.432  3rd Qu.: 8.3001  3rd Qu.:15.000
Max.   :135.019  Max.   :1136.495  Max.   :19.8013  Max.   :19.000
```

Added to the minimum, maximum and mean of any data column, we can read the first quartile, median and third quartile. In general these statistics can be informative on the distribution of each quantity. For instance, the fact that first quartile and median are much closer to the minimum than to the mean for IMEAN tells that the underlying distribution is highly skewed.

What is missing from the above analysis is resolution, because this is not included in data.frame “mtzdata”. A resolution column needs, thus, to be added to this data.frame. To calculate the resolution *d* for each reflection the following well-known formula can be used:

$$1/d^2 = T/B$$

where,

$$T = 1 - \cos^2\alpha - \cos^2\beta - \cos^2\gamma + 2\cos\alpha\cos\beta\cos\gamma$$

and,

$$B = \frac{h^2}{a^2}\sin^2\alpha + \frac{k^2}{b^2}\sin^2\beta + \frac{l^2}{c^2}\sin^2\gamma + 2hk\left(\cos\alpha\cos\beta - \cos\gamma\right)$$
$$+ 2hl\left(\cos\alpha\cos\gamma - \cos\beta\right) + 2kl\left(\cos\beta\cos\gamma - \cos\alpha\right)$$

Of course this formula is not readily available in R; it needs to be defined as an R-function. The function, named “d_hkl”, reads in Miller indices and cell parameters, and returns the corresponding resolution in angstroms. The function can be typed in as follows:

```
d_hkl <- function(h,k,l,a,b,c,aa,bb,cc)
{
 # Given Miller indices and cell parameters, this function returns
 # resolution corresponding to the specific Miller indices.

 aa <- aa*pi/180
 bb <- bb*pi/180
 cc <- cc*pi/180
 top <- 1-(cos(aa))^2-(cos(bb))^2-(cos(cc))^2+2*cos(aa)*cos(bb)*cos(cc)
 b1 <- h^2*(sin(aa))^2/a^2
 b2 <- k^2*(sin(bb))^2/b^2
 b3 <- l^2*(sin(cc))^2/c^2
 b4 <- 2*h*k*(cos(aa)*cos(bb)-cos(cc))/(a*b)
 b5 <- 2*h*l*(cos(aa)*cos(cc)-cos(bb))/(a*c)
 b6 <- 2*k*l*(cos(bb)*cos(cc)-cos(aa))/(b*c)
 d2 <- top/(b1+b2+b3+b4+b5+b6)
 return(sqrt(d2))
}
> class(d_hkl)
[1] "function"
```

The application of this function to all Miller indices of data.frame “mtzdata” shows once more the ability and convenience of using R to carry out a same operation in parallel fashion:

```
> resos <- d_hkl(mtzdata$H,mtzdata$K,mtzdata$L,77.63,77.63,77.63,90,90,90)
> length(resos)
[1] 7007
> summary(resos)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.823   2.018   2.313   2.768   2.930  27.450
```

In the R code just typed “resos” is a new object vector of length 7007 containing resolutions for all Miller indices of the “mtzdata” data.frame, following its same ordering. “resos” stores the parallel application to all triples h, k, l of the function “d_hkl”. This function takes in only one value for h, one value for k and one value for l. Given that the expression “mtzdata$H” and similar expressions for K and L contains all 7007 values for h, k and l, R is clever enough to work out that the operations within function “d_hkl” have to be carried out in vectorised fashion across all triples h, k, l, keeping fixed cell parameters. Finally, “summary” returns resolutions statistics. From the result it is easy to check that minimum and maximum resolutions coincides with that shown in Figure 4.

4. Non-standard analysis of an mtz reflection file

The availability of data in data.frame format enables us to carry out non-standard analysis for the kind of data available in an mtz file. Let us consider, for instance, the issue with FREERFLAG. As it is known, each reflection of an mtz file is tagged with a random integer number in a given range. This has the purpose of dividing data in smaller, random sets to be used, for instance, for cross validation. Independent sampling guarantees that any observed reflection has the same probability to be selected for any given statistical operation. We could, for instance, decide to test the distribution of flags across reflections to make sure that specific integer values are uniformly spread across all resolutions.

Having mtz data available in a data.frame within R allow them to be easily separated using conditional statements. For example, to extract from “mtzdata” all reflections with free R flag equal to 10, we simply type (in the mtz file used here the free R flag has all integer numbers between 0 and 19):

```
> data10 <- mtzdata[mtzdata$FreeR_flag == 10,]
> str(data10)
'data.frame': 336 obs. of  8 variables:
 $ H         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ K         : int  3 4 5 5 6 10 11 12 12 13 ...
 $ L         : int  15 30 5 33 4 28 37 0 28 5 ...
 $ IMEAN     : num  249.8 158.9 92.5 21 22.5 ...
 $ SIGIMEAN  : num  4.86 7.31 3.62 6.89 1.42 ...
 $ F         : num  158 125.9 96.1 43 47.3 ...
 $ SIGF      : num  1.54 2.91 1.89 9.12 1.51 ...
 $ FreeR_flag: int  10 10 10 10 10 10 10 10 10 10 …
```

The new data.frame obtained, which we named “data10”, is composed, of 336 reflections whose “FreeR_flag” column only displays value 10, as it should be. Are quantities included in this data.frame statistically different from the others? For example, if we selected a different data.frame using a free R flag equal to 5, would we notice any major difference with data in “data10”? The answer is, obviously, no. To see this let us, first, create the new data.frame, which we call “data05”:

```
> str(data05)
'data.frame': 333 obs. of  8 variables:
 $ H         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ K         : int  7 7 7 12 12 13 15 16 18 18 ...
 $ L         : int  9 21 35 4 16 15 25 12 12 26 ...
 $ IMEAN     : num  1495 255 1017 1260 157 ...
 $ SIGIMEAN  : num  22.98 7.39 14.76 11.23 16.91 ...
 $ F         : num  387 160 319 355 125 ...
 $ SIGF      : num  2.97 2.32 2.32 1.58 6.81 ...
 $ FreeR_flag: int  5 5 5 5 5 5 5 5 5 5 …
```

Now let us compute summary statistics, for some of the observations, for example for the structure factors amplitudes:

```
> summary(data10$F)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12.42   40.58   79.25  121.90  133.10  943.90
> summary(data05$F)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.98   43.89   90.23  123.00  156.10  599.10
```

There seem to be no major differences between the two sets of reflections. The maximum is larger for structure factors related to “data10”, but this can normally happen because individual structure factors carry atomic structural information of differing strengths at different resolutions. The important fact to register in this comparison is that all other measures of statistical tendency are pretty close to each other. Histograms for the two distributions can be easily calculated and illustrated using the “hist” function (to find out about the various parameters used in this example, please refer to the manual pages, “help(hist)”):

```
> countsA <- hist(data10$F,breaks=seq(0,1000,length=11),main="Comparison of
frequencies",xlab=expression(list(F[A],F[B])),ylab="Frequency")
> countsB <- hist(data05$F,breaks=seq(0,1000,length=11),lty=2,add=TRUE)
```

```
> legend(600,170,legend=c(expression(F[A]),expression(F[B])),lty=c(1,2))
> countsA <- countsA$counts
> countsA
 [1] 210  69  28  11   6   6   3   1   1   1
> countsB < - countsB$counts
> countsB
 [1] 189  82  32  18   7   5   0   0   0   0
```

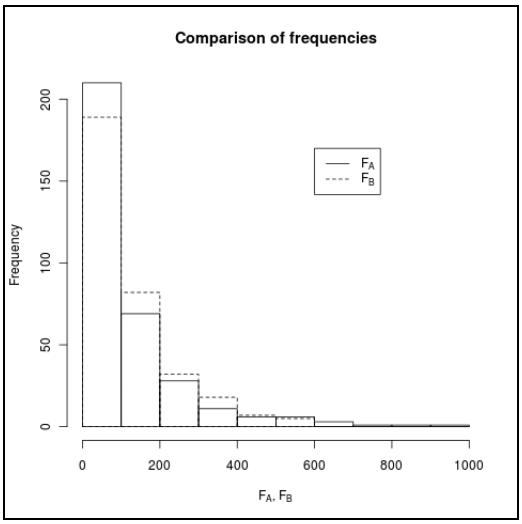


Figure 5

In Figure 5 F_A are structure factors in “data10” and F_B those in “data05”. The two distributions seem very similar. A somewhat conclusive proof that they do not differ significantly can be provided by some statistical test. We can, for example, use a well-established non-parametric test known as *Wilcoxon test* [9]. This is normally used to estimate whether two sets of numbers differ significantly or not. In our case the two sets are the number of structure factors amplitudes within each bin in the histogram. As with many other tests, the Wilcoxon test calculates a so-called *p value* to estimate confidence probabilities. The standard confidence used is the 95% one. This means that if the p value is less than 0.05 there is not enough statistical ground to consider the two sets as having the same distribution, and the null hypothesis of considering them equal is rejected. In R the test can be performed with just one command:

```
> wilcox.test(countsA,countsB)

        Wilcoxon rank sum test with continuity correction

data:  countsA and countsB
W = 57, p-value = 0.6212
alternative hypothesis: true location shift is not equal to 0

Warning message:
In wilcox.test.default(countsA, countsB) :
```

```
cannot compute exact p-value with ties
```

As the p value is well above 0.05 we can be confident that the two different groups of structure factors follow a same distribution (ignore the warning message; this is simply telling that four of the differences are equal and will be assigned equal ranking values). This implies that the free R flag assignment does not produce any statistically meaningful difference among reflections.

5. Conclusions

The use of statistical packages is as important in crystallographic research as it is in any other scientific field. Data analysis forms the basis for quantitative investigations. It is, therefore, not surprising that statistics should be used appropriately with any scientific finding. A special session was devoted to statistical packages at the European Crystallographic Meeting in 2012 (Bergen, Norway). The first two presentations of that session focussed on the R package and on how it could be used to help out in crystallographic investigations [10, 11]. Other authors have used the package and are still using it in their research. It is foreseeable that use of R will increase in crystallography, given the very solid foundations on which it has been built and the impressive speed of diffusion among scientists and academic workers in general.

In this short article it has been shown that it is possible to use of R for statistical data analysis of crystallographic files, without resorting to the standard tools available within the CCP4 suite. Results already obtained with the CCP4 software can be easily reproduced. This provides a feasible path to control and reinforce calculations. Furthermore new types of analysis can be planned and carried out using the huge arsenal of R built-in functions.

Many crystallographic common operations, though, necessitate of specific algorithms. In general these are not implemented either in the R core on in contributed R software. There exist, accordingly, some limitations in the variety of possible crystallographic investigations within R. Additional software, possibly in the form of R packages, is needed to overcome this limitation. Promising steps in this direction have been taken with the creation of two projects dealing with diffraction images [8] and general crystallographic operations [7]. Use of the new software associated with these projects increases substantially the extent of crystallographic analysis.

Acknowledgments

The author would like to thank Andrey Lebedev for reading the manuscript and for providing useful suggestions to improve it.

References

[1] IBM SPSS Statistics 21.0 - August 2012. http://www-01.ibm.com/software/analytics/spss/

[2] StataCorp. 2011. *Stata Statistical Software: Release 12*. College Station, TX, USA: StataCorp LP. http://www.stata.com/

[3] Minitab 16 Statistical Software (2010). [Computer software]. State College, PA, USA: Minitab, Inc. http://www.minitab.com/en-US/default.aspx

[4] R Core Team, R: A Language and Environment for Statistical Computing (2013) – R Foundation for Statistical Computing - Vienna, Austria. http://www.R-project.org

[5] S-PLUS. TIBCO Software Inc. (2010). Palo Alto, CA, USA. http://www.tibco.com

[6] http://journals.iucr.org

[7] http://code.google.com/p/cry-package

[8] http://code.google.com/p/disp/

[9] G. W. Corder and D. I. Foreman (2009), *Nonparametric Statistics for Non-Statisticians*, Wiley

[10] G. N. Murshudov (2012), Statistics package *R for prototyping in macromolecular crystallography*, Acta Cryst. A**68**, s79

[11] M. W. Baumstark (2012), *R as a tool to check data quality in the context of low-resolution crystallography*, Acta Cryst. A**68**, s80