# Online Appendix 5.A: Calculating the Wilcoxon statistic and showing its equivalence to area under empirical ROC curve

File mainWilcoxon.R illustrates calculation of the empirical AUC using the Wilcoxon statistic.

### Online Appendix 5.A.1: Code Listing

rm( list = ls()) # mainWilcoxon.R

library(caTools)

source("Wilcoxon.R");source("RocOperatingPoints.R")

options(digits = 9)

RocCountsTable = array(dim = c(2,5))

RocCountsTable[1,] <- c(30,19,8,2,1)

RocCountsTable[2,] <- c(5,6,5,12,22)

#convert frequency table to array

zk1 <- rep(1:length(RocCountsTable[1,]),

RocCountsTable[1,])

zk2 <- rep(1:length(RocCountsTable[2,]),

RocCountsTable[2,])

w <- Wilcoxon (zk1, zk2)

cat("The wilcoxon statistic is = ", w, "\n")

ret <- RocOperatingPoints(RocCountsTable[1,],

RocCountsTable[2,])

FPF <- ret$FPF;FPF <- c(0,FPF,1)

TPF <- ret$TPF;TPF <- c(0,TPF,1)

AUC <- trapz(FPF,TPF) # trapezoidal integration

cat("direct integration yields AUC = ", AUC, "\n")

The function that calculates the Wilcoxon statistic is in file Wilcoxon.R.

### Online Appendix 5.A.2: Code Listing

Wilcoxon <- function (zk1, zk2)

{

K1 = length(zk1)

K2 = length(zk2)

W <- 0

for (k1 in 1:K1) {

W <- W + sum(zk1[k1] < zk2)

W <- W + 0.5 \* sum(zk1[k1] == zk2)

}

W <- W/K1/K2

return (W)

}

Insert a breakpoint (red dot) at line 15 in file mainWilcoxon.R, Figure 1, and Source the file.

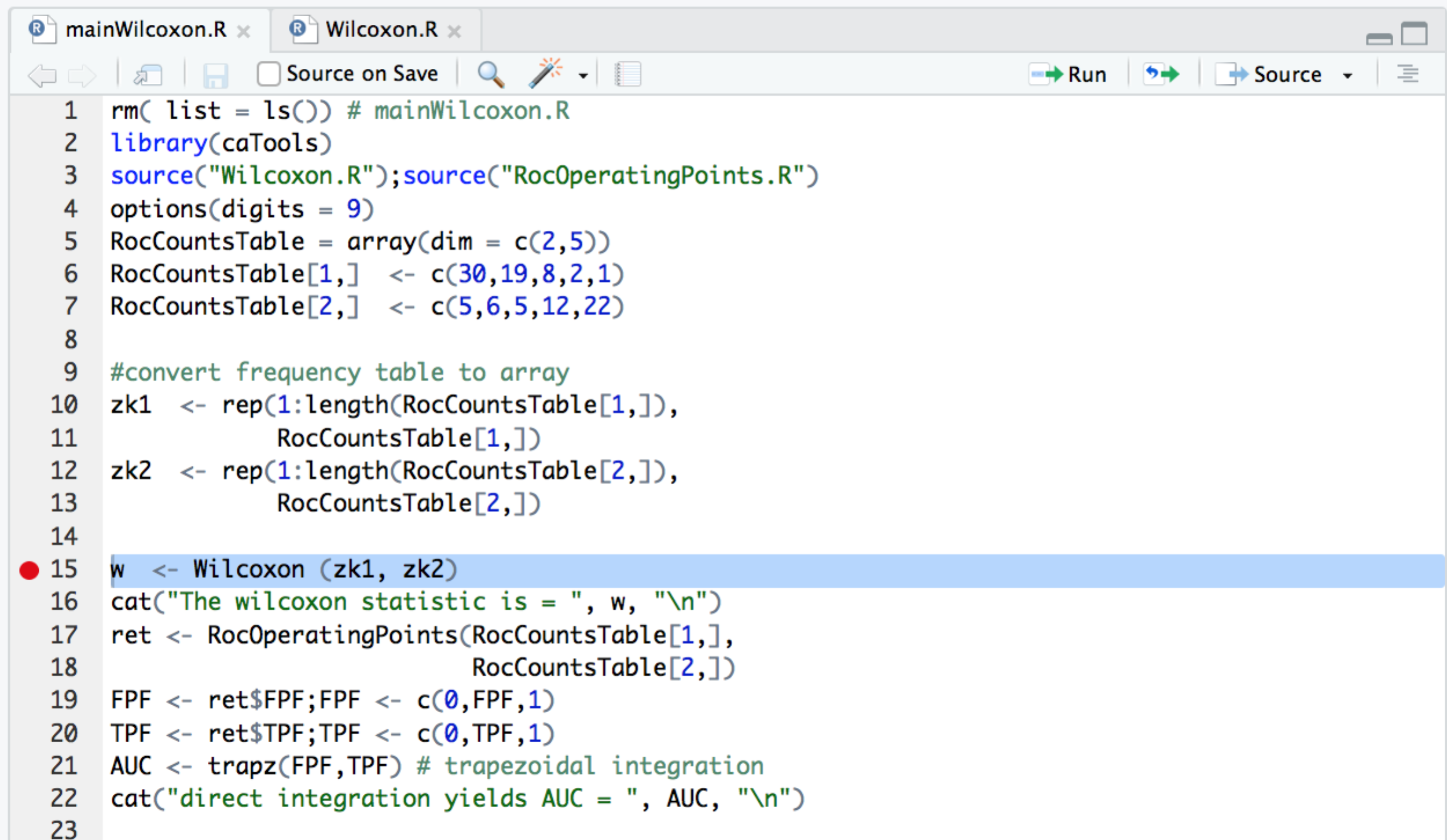


Figure 1: The red dot indicates that a break has been inserted at line 12. Sourcing the file cause execution to suspend in debug mode at this line, and Debug options are shown as in Figure 2.

In the Console window one should see Figure 2.

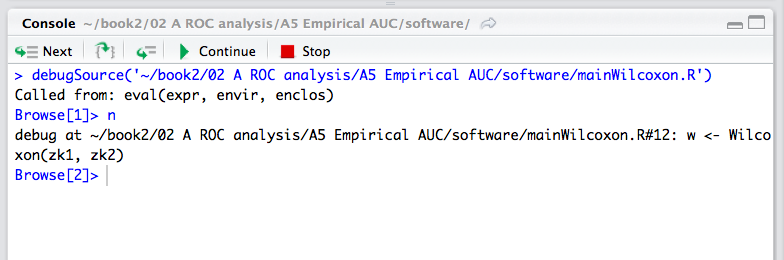


Figure 2: This screen-shot of the Console windows shows the debug options; the buttons are from left to right: Next, "*enter function/loop*", "*exit function loop*", Continue execution and Stop or exit debug mode.

Clicking on Next executes the statement, but don't do so right now. The symbol to the immediate right of the Next button, which could be described as an arrow stepping into code contained in braces is what the author terms "*enter loop/function*". The next-right button steps out of a function (or from inside a for-loop) to the calling statement or the outer layer, is what the author terms "*exit loop/function*". The button labeled Continue executes all code until the next break point is encountered, if any, and finally the Stop button gets one out of debug mode. Click the "*enter loop/function*" button. The code pointer should be at line 1 in file Wilcoxon.R, Figure 3.

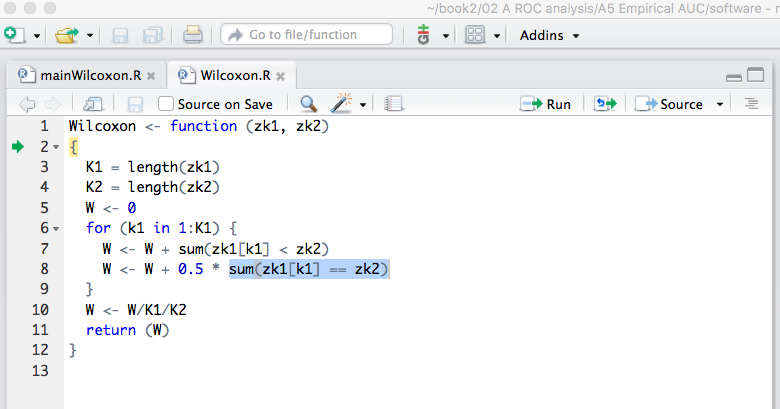


Figure : Effect on entering function; code pointer is at line 1.

The green arrow shows the next statement to be executed and new debug menu items have appeared in the Console window. Lines 3 and 4 calculate the number of non-diseased K1 and diseased cases K2, respectively. Keep clicking Next until the code pointer is at line 7.

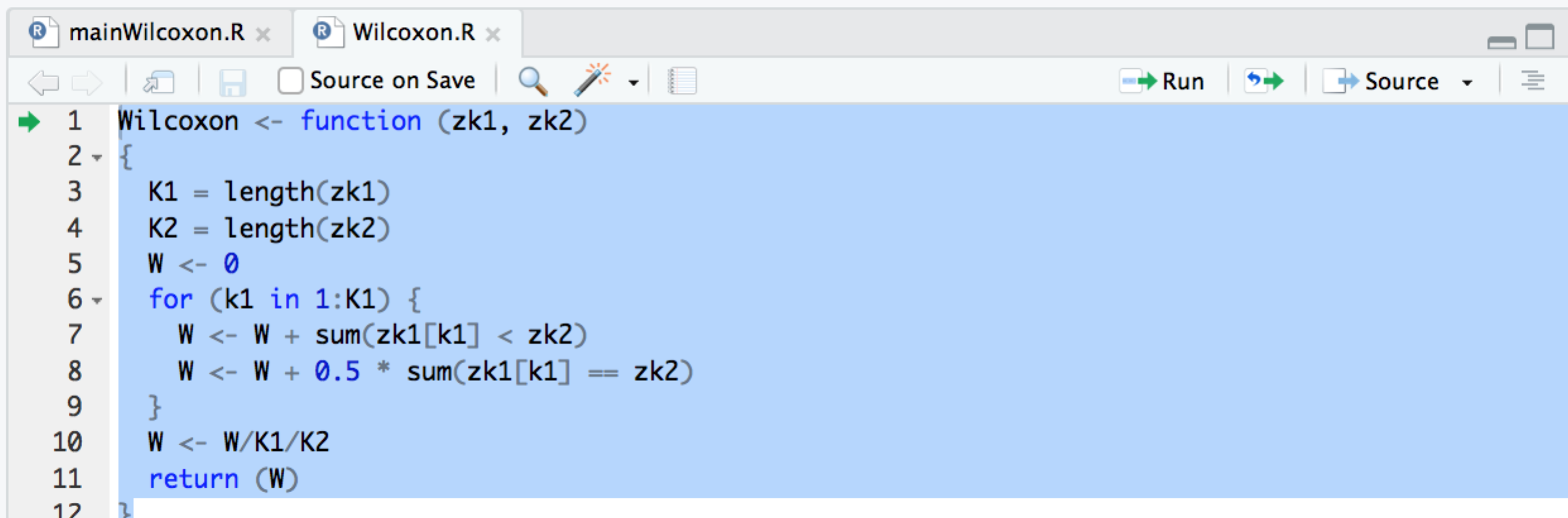


Figure 4: Code pointer has been advanced to line 7 by repeatedly clicking Next.

The construct sum(zk1[k1] < zk2) sums the number of instances that a specific value of z[k1] is smaller than elements in zk2. The next line sums the number of instances that a specific value of z[k1] equals elements in zk2. The illustration below gives the general idea.

### Online Appendix 5.A.3: Code Snippet

Browse[3]> zk2

[1] 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5

[41] 5 5 5 5 5 5 5 5 5 5

Browse[3]> zk1[k1]

[1] 1

Browse[3]> zk1[k1] < zk2

[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

[14] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

[27] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

[40] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

Browse[3]> sum(zk1[k1] < zk2)

[1] 45

Browse[3]> sum(zk1[k1] == zk2)

[1] 5

There are 45 instances of the first non-diseased case z-sample being smaller than any diseased case z-sample and 5 instances where the two are equal. Clicking Next twice gets to the next value of k1, and W is 47.5 (confirm in Environment window). Click the "*get me out of this loop/function*" button, which keeps repeating lines 7 and 8 for the rest of the non-diseased cases. The final result is W is 2582, which is obviously not a probability. Dividing by the total number of comparisons yields the final value of the Wilcoxon statistic: click Next, revealing W = 0.8607, which is the desired Wilcoxon statistic. Click "*get me out of this loop/function*" to return to line 13 of the main function, Figure 5.

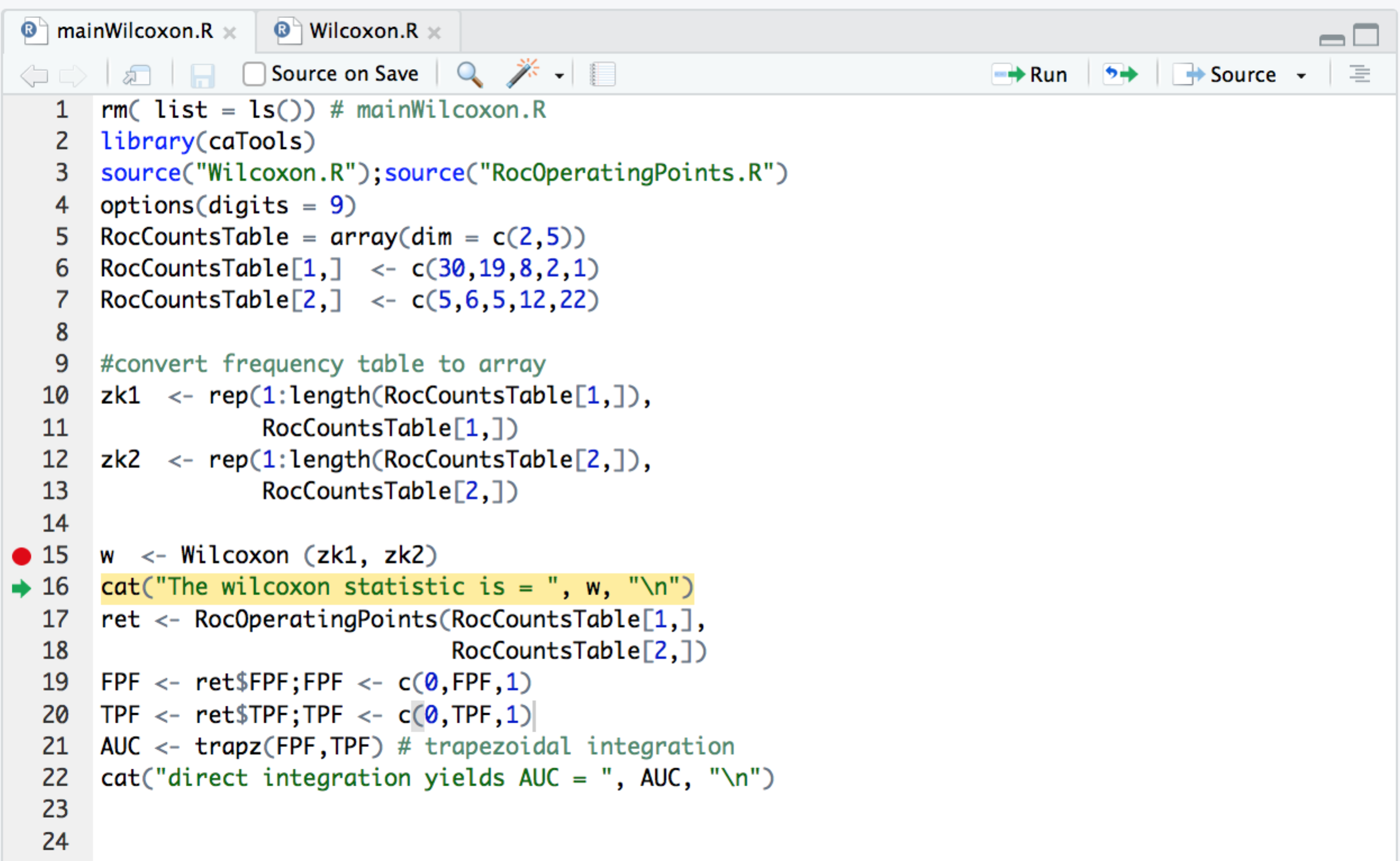


Figure 5: Back in main function.

Line 17 calculates the operating points corresponding to the data in RocCountsTable; the author leaves it to the reader to enter this function to see the formulae in book Eqn. 5.9 implemented. The returned variable contains FPF and TPF as elements of a list variable; these are extracted at lines 19-20; line 21 performs the trapezoidal integration using the function trapz() contained in package caTools. The rest of the code is cat() statements that print values.

Remove break points, exit debug mode and source the code to get the following output.

> source(…)

The wilcoxon statistic is = 0.860666667

direct integration yields AUC = 0.860666667

# Online Appendix 5.B: Plotting the empirical ROC and shading it

File mainEmpiricalAUC.R implements this. Source it to get book Figure 5.2. Documenting and / or simplification of this code are left to a user.

# Online Appendix 5.C: The Bamber theorem plot

File MainBamberTheorem.R illustrates this. Source it to get book Figure 5.3. Documenting and / or simplification of this code are left to a user.

# Online Appendix 5.D: Empirical AUC illustrating the numbering of the operating points

File mainEmpRocPlot.R illustrates this. Source it to get book Figure 5.1. Documenting and / or simplification of this code are left to a user.