Online Appendix 5.A: Calculating the Wilcoxon statistic

The 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")

RocCountsTable = array(dim = c(2,5))
RocCountsTable[1,]  <- c(30,19,8,2,1)
RocCountsTable[2,]  <- c(5,6,5,12,22)

zk1  <- rep(1:length(RocCountsTable[1,]),RocCountsTable[1,])#convert frequency table to array
zk2  <- rep(1:length(RocCountsTable[2,]),RocCountsTable[2,])#do:

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

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

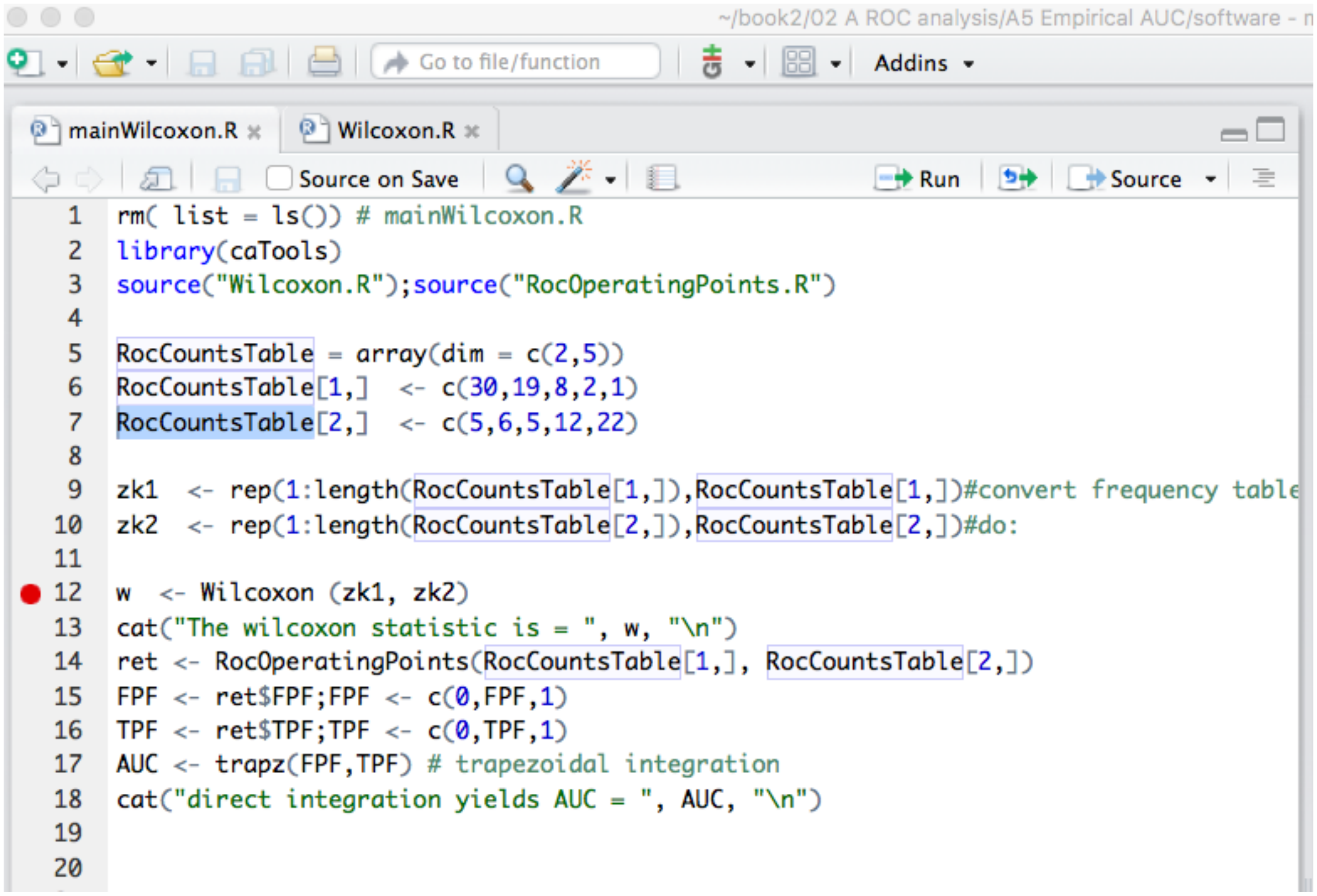
Online Appendix 5.A.2: Code Listing

```
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)
}
```

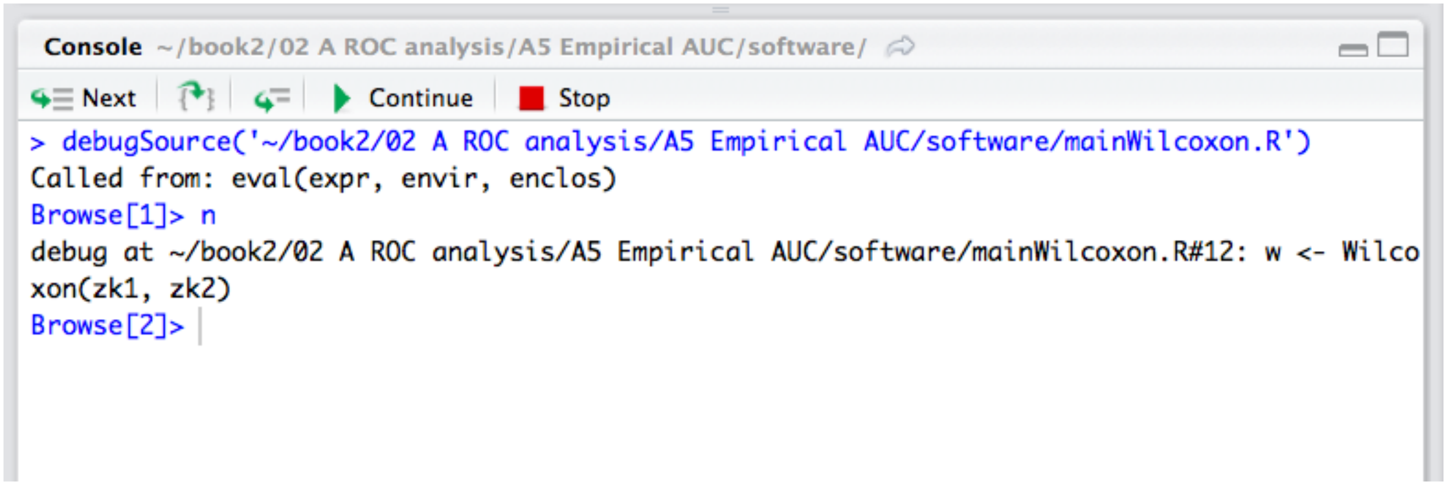
Insert a breakpoint (red dot) at line 12 in file **mainWilcoxon.R**, Fig. 5.A.1, and **Source** the file.



```
 1  rm( list = ls()) # mainWilcoxon.R
 2  library(caTools)
 3  source("Wilcoxon.R");source("RocOperatingPoints.R")
 4
 5  RocCountsTable = array(dim = c(2,5))
 6  RocCountsTable[1,]  <- c(30,19,8,2,1)
 7  RocCountsTable[2,]  <- c(5,6,5,12,22)
 8
 9  zk1  <- rep(1:length(RocCountsTable[1,]),RocCountsTable[1,])#convert frequency table
10  zk2  <- rep(1:length(RocCountsTable[2,]),RocCountsTable[2,])#do:
11
12  w  <- Wilcoxon (zk1, zk2)
13  cat("The wilcoxon statistic is = ", w, "\n")
14  ret <- RocOperatingPoints(RocCountsTable[1,], RocCountsTable[2,])
15  FPF <- ret$FPF;FPF <- c(0,FPF,1)
16  TPF <- ret$TPF;TPF <- c(0,TPF,1)
17  AUC <- trapz(FPF,TPF) # trapezoidal integration
18  cat("direct integration yields AUC = ", AUC, "\n")
19
20
```

Fig. 5.A.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, see Fig. 5.A.2.

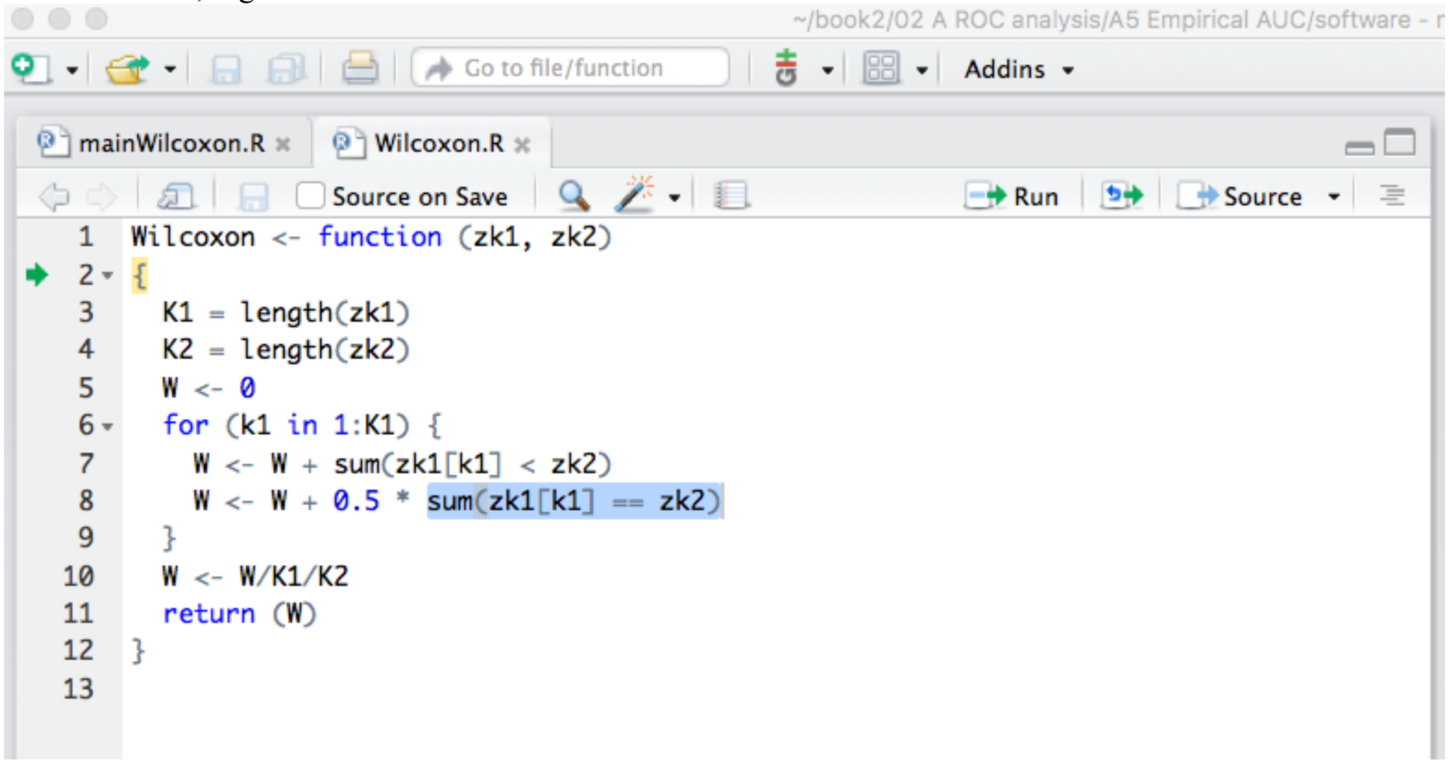
In the **Console** window one should see Fig. 5.A.2.


Console ~/book2/02 A ROC analysis/A5 Empirical AUC/software/
Next
Continue
Stop

```
> debugSource('~/book2/02 A ROC analysis/A5 Empirical AUC/software/mainWilcoxon.R')
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at ~/book2/02 A ROC analysis/A5 Empirical AUC/software/mainWilcoxon.R#12: w <- Wilco
xon(zk1, zk2)
Browse[2]>
```

Fig. 5.A.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.

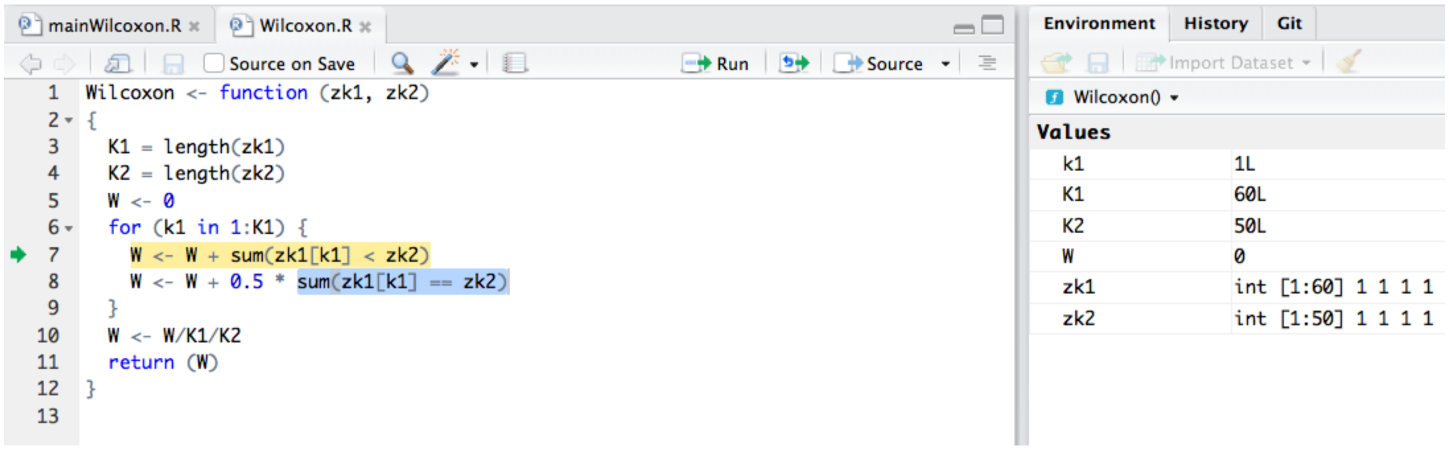
Clicking on **Next** would simply execute the statement. The symbol to its immediate right, which could be described as an arrow stepping into code contained in braces [1] is what the author terms "*enter nearest loop/function*". The next right button steps out of a function (or from inside a for-loop) to the calling statement or the outer layer, is what the author terms "*exit nearest 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, Fig. 5.A.2. The code pointer should be at line 2 in file **Wilcoxon.R**, Fig. 5.A.3.



```
 1   Wilcoxon <- function (zk1, zk2)
 2   {
 3     K1 = length(zk1)
 4     K2 = length(zk2)
 5     W <- 0
 6     for (k1 in 1:K1) {
 7       W <- W + sum(zk1[k1] < zk2)
 8       W <- W + 0.5 * sum(zk1[k1] == zk2)
 9     }
10     W <- W/K1/K2
11     return (W)
12   }
13
```

Fig. 5.A.3: Effect on entering function; code pointer is at line 2.

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 extract the number of non-diseased **K1** and diseased cases **K2**, respectively. Keep clicking **Next** until the code pointer is at line 7.



```
 1   Wilcoxon <- function (zk1, zk2)
 2   {
 3     K1 = length(zk1)
 4     K2 = length(zk2)
 5     W <- 0
 6     for (k1 in 1:K1) {
 7       W <- W + sum(zk1[k1] < zk2)
 8       W <- W + 0.5 * sum(zk1[k1] == zk2)
 9     }
10     W <- W/K1/K2
11     return (W)
12   }
13
```

Values

k1	1L
K1	60L
K2	50L
W	0
zk1	int [1:60] 1 1 1 1
zk2	int [1:50] 1 1 1 1

Fig. 5.A.4: Code pointer has been advanced to line 7 by repeatedly clicking **Next**.

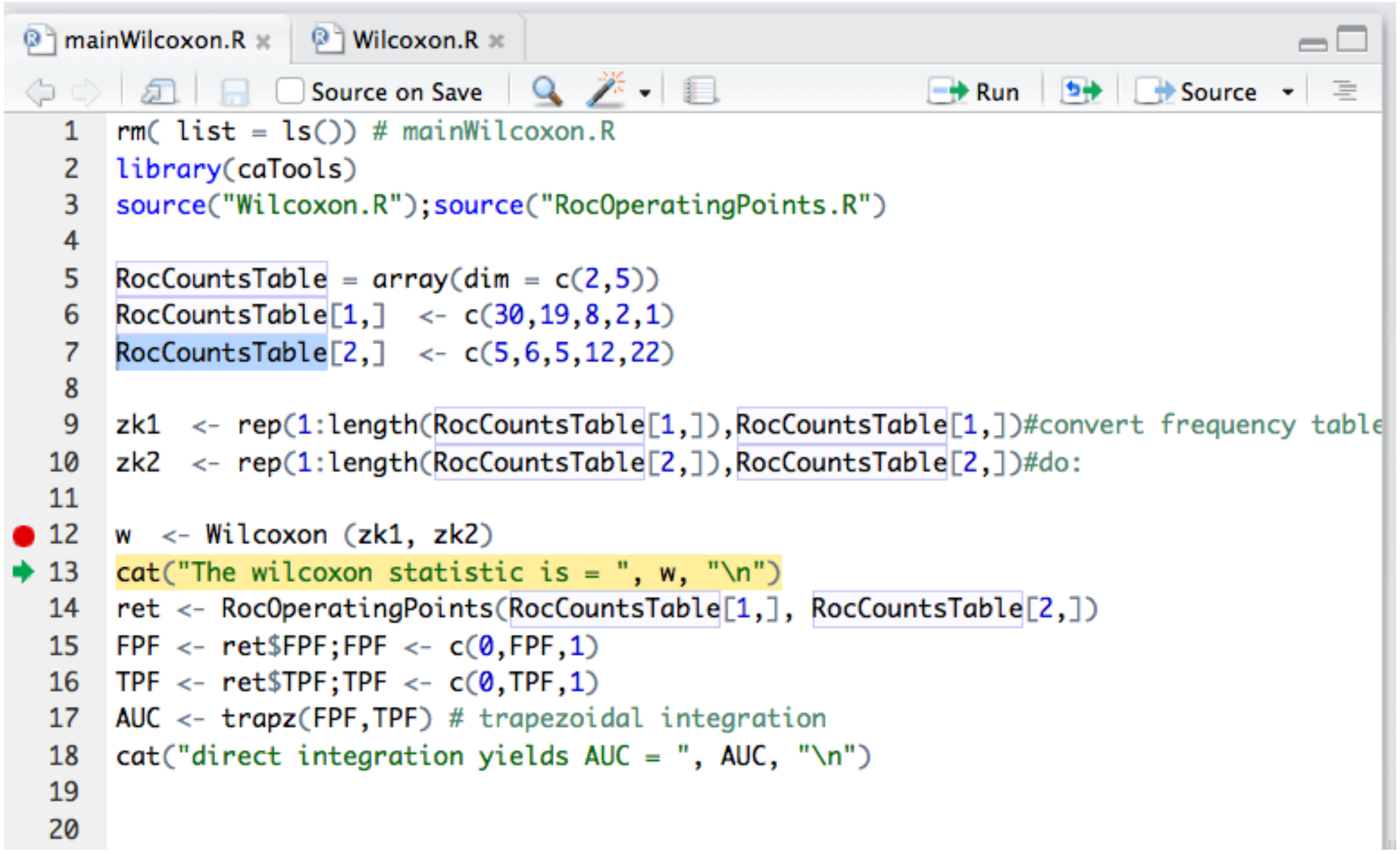
The construct **sum(zk1[k1] < zk2)** sums the number of instances that a specific value of **z[k1]** is smaller than elements in **zk2**. The next line sums the number of instances that a specific value of **z[k1]** equals elements in **zk2**. The illustration below gives the general idea.

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. 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, Fig. 5.A.5.



```
 1  rm( list = ls()) # mainWilcoxon.R
 2  library(caTools)
 3  source("Wilcoxon.R");source("RocOperatingPoints.R")
 4
 5  RocCountsTable = array(dim = c(2,5))
 6  RocCountsTable[1,]  <- c(30,19,8,2,1)
 7  RocCountsTable[2,]  <- c(5,6,5,12,22)
 8
 9  zk1  <- rep(1:length(RocCountsTable[1,]),RocCountsTable[1,])#convert frequency table
10  zk2  <- rep(1:length(RocCountsTable[2,]),RocCountsTable[2,])#do:
11
12  w  <- Wilcoxon (zk1, zk2)
13  cat("The wilcoxon statistic is = ", w, "\n")
14  ret <- RocOperatingPoints(RocCountsTable[1,], RocCountsTable[2,])
15  FPF <- ret$FPF;FPF <- c(0,FPF,1)
16  TPF <- ret$TPF;TPF <- c(0,TPF,1)
17  AUC <- trapz(FPF,TPF) # trapezoidal integration
18  cat("direct integration yields AUC = ", AUC, "\n")
19
20
```

Fig. 5.A.5: Back in main function.

Line 14 calculates the operating points corresponding to the data in **RocCountsTable**; the author leaves it to the reader to enter this function to seeing the formulae in Eqn. **Error! Reference source not found.** implemented. The returned variable contains FPF and TPF as elements of a **list** variable; these are extracted at lines 15-16; line 17 performs the trapezoidal integration using the function **trapz()** contained in package **caTools**. The rest of the code is print statements.