# *The Art of R Programming*

## A Tour of Statistical Software Design

by Norman Matloff

errata updated to print 14

| Page | Error | Correction | Print corrected |
|---|---|---|---|
| 1 | If not, see Appendix A.<br>**for installation instructions.** | If not, see Appendix A **for installation instructions.** | Print 3 |
| 51 | `vud <- diff(d)` | `vud <- diff(v)` | Print 3 |
| 53 | `for (gen in c("M","F")) grps[[gen]] <- which(aba==gen)` | `for (gen in c("M","F")) grps[[gen]] <- which(aba[,1]==gen)` | Print 2 |
| 65, 66 | `newimg@grey <- (1-q) * img@grey + q * randomnoise` | `newimg@grey[rows,cols] <- (1-q) * img@grey[rows,cols] + q * randomnoise` | Print 4 |
| 67 | `> z <- c(5,12,13)`<br>`> x[z %% 2 == 1,]`<br>`[,1] [,2]` | `> x[z %% 2 == 1,]`<br>`[,1] [,2]` | Print 3 |
| 68 | `[1,]    1    4`<br>`[2,]    3    6` | `[1,]    1    2`<br>`[2,]    3    4` | Print 3 |
| 77 | Recall that due to the symmetry of the matrix, we skip the early part of each row, as is seen in the expression `(i+1):(1x-1)` in line 18. But that means that the call to `which.min()` in that line will return the minimum's index *relative* to the range `(i+1):(1x-1)`. | Recall that due to the symmetry of the matrix, we skip the early part of each row, as is seen in the expression `(i+1):(lx-1)` in line 18. But that means that the call to `which.min()` in that line will return the minimum's index *relative* to the range `(i+1):(lx-1)`. | Print 3 |
| 93 | `> nwords <- length(ssnyt)`<br>`> barplot(freqs9)` | `> nwords <- length(ssnyt)`<br>`> freqs9 <- sapply(ssnyt[round(0.9*nwords):nwords],length)`<br>`> barplot(freqs9)` | Print 2 |

| Page | Error | Correction | Print corrected |
|---|---|---|---|
| 116 | `shang3` | `shang4` | Print 2 |
| 128 | `fl.1 a bc`<br>`  5 2 1`<br>`  12 1 1`<br>`  13 1 0` | `fl.1 a bc`<br>`  5 2 0`<br>`  12 1 1`<br>`  13 2 1` | Print 2 |
| 130 | `> ctt/5` | `> cttab/5` | Print 2 |
| 131 | `> apply(ctt,1,sum)` | `> apply(cttab,1,sum)` | Print 2 |
| 133 | `f(argslist[[1],argslist[[2]],...)` | `f(argslist[[1]],argslist[[2]],...)` | Print 2 |
| 137 | This says that z[1], 0.88114802, fell into bin 9, which was (0,**0**,0.**1**]; z[2], 0.28532689, fell into bin 3; and so on. | This says that z[1], 0.88114802, fell into bin 9, which was (0.**8**,0.**9**]; z[2], 0.28532689, fell into bin 3; and so on. | Print 2 |
| 148 | Good software design, however, should **be** mean that you can glance through a function's code . . . | Good software design, however, should mean that you can glance through a function's code . . . | Print 3 |
| 151 | `> f(3,2)`<br>`[1] 1`<br>`> g <- function(h,a,b) h(a,b)`<br>`> g(f1,3,2)`<br>`[1] 5`<br>`> g(f2,3,2)`<br>`[1] 1` | `> f(3,2)`<br>`[1] 1`<br>`> g <- function(x) x^2`<br>`> body(g) <- quote(2*x+3)`<br>`> g`<br>`function (x)`<br>`2 * x + 3`<br>`> g(8)`<br>`[1] 19` | Print 5 |

| Page | Error | Correction | Print corrected |
|---|---|---|---|
| 151 | `> g <- function(h,a,b) h(a,b)`<br>`> body(g) <- quote(2*x + 3)`<br>`> g`<br>`function (x)`<br>`2 * x + 3`<br>`> g(3)`<br>`[1] 9` | `> g <- function(h,a,b) h(a,b)`<br>`> body(g) <- quote(2*x + 3)`<br>`> g`<br>`function (x)`<br>`2 * x + 3`<br>`> x <- 3`<br>`> g(3)`<br>`[1] 9` | Print 2 |
| 155 | `> f(2)`<br>`[1] 88` | `> f(2)`<br>`[1] 112` | Print 2 |
| 160 | `> oddsevens`<br>`function(v){`<br>`    odds <- which(v %% 2 == 1)`<br>`    evens <- which(v %% 2 == 1)`<br>`    list(o=odds,e=evens)`<br>`}` | `> oddsevens`<br>`function(v){`<br>`    odds <- which(v %% 2 == 1)`<br>`    evens <- which(v %% 2 == 0)`<br>`    list(o=odds,e=evens)`<br>`}` | Print 2 |
| 163 | `makecorpdfs(c("MICROSOFT CORPORATION","ms","INTEL CORPORATION","intel","`<br>`SUN MICROSYSTEMS, INC.","sun","GOOGLE INC.","google")` | `makecorpdfs(c("MICROSOFT CORPORATION","ms","INTEL CORPORATION","intel","`<br>`SUN MICROSYSTEMS, INC.","sun","GOOGLE INC.","google"))` | Print 2 |
| 164 | . . . when we discuss appropriate use global variables in the next section. | . . . when we discuss appropriate use **of** global variables in the next section. | Print 3 |
| 176 | 3. Within `f()`, piece together the results of (**b**) to solve the original problem. | 3. Within `f()`, piece together the results of (**2**) to solve the original problem. | Print 3 |
| 178 | . . . while the right subtree stores the elements that are larger than the value in this **mode**. | . . . while the right subtree stores the elements that are larger than the value in this **node**. | Print 3 |

| Page | Error | Correction | Print corrected |
|---|---|---|---|
| 185 | 26 \end{Code}<br>27<br>28 Let's test it.<br>29<br>30 \begin{Code}<br>31 > b <- newbookvec(c(3,4,5,5,12,13))<br>32 > b<br>33 $vec<br>34 [1] 3 4 5 5 12 13<br>35<br>36 $wrts<br>37 [1] 0 0 0 0 0 0<br>38<br>39 attr(,"class")<br>40 [1] "bookvec"<br>41 > b[2]<br>42 [1] 4<br>43 > b[2] <- 88 # try writing<br>44 > b[2] # worked?<br>45 [1] 88<br>46 > b$wrts # write count incremented?<br>47 [1] 0 1 0 0 0 0 | **Let's test it.**<br><br>> b <- newbookvec(c(3,4,5,5,12,13))<br>> b<br>$vec<br>[1] 3 4 5 5 12 13<br>$wrts<br>[1] 0 0 0 0 0 0<br>attr(,"class")<br>[1] "bookvec"<br>> b[2]<br>[1] 4<br>> b[2] <- 88 # try writing<br>> b[2] # worked?<br>[1] 88<br>> b$wrts # write count incremented?<br>[1] 0 1 0 0 0 0 | Print 3 |
| 191 | The expression `notp[-i]` computes the product of all the elements of notp, . . . | The expression **`prod(`**`notp[-i]`**`)`** computes the product of all the elements of notp, . . . | Print 3 |
| 194 | *For instance, to find* ***our*** *more about the chi-square function for quantiles, . . .* | *For instance, to find* ***out*** *more about the chi-square function for quantiles, . . .* | Print 3 |
| 197 | > a <- matrix(c(1,1,-1,1),nrow=2,ncol=2)<br>> b <- c(2,4)<br>> solve(a,b)<br>[1] 3 1<br>> solve(a)<br>  [,1] [,2]<br>[1,] 0.5 0.5<br>[2,] -0.5 0.5 | > a <- matrix(c(1,-1,1,1),nrow=2)<br>> b <- c(2,4)<br>> solve(a,b)<br>[1] -1 3<br>> solve(a)<br>    [,1] [,2]<br>[1,] 0.5 -0.5<br>[2,] 0.5 0.5 | Print 3 |
| 206 | Recalling that R lists are often used to store several related variables in one basket, we **se** up a list `comdat`. | Recalling that R lists are often used to store several related variables in one basket, we **set** up a list `comdat`. | Print 3 |
| 228 | > save(hz,"hzfile") | > save(hz,**file=**"hzfile") | Print 3 |

| Page | Error | Correction | Print corrected |
|---|---|---|---|
| 264 | • On a Mac, call `macintosh()`. | • On a Mac, call `quartz()`. | Print 3 |
| 276 | `g <- function(t) { return (t^2+1)^0.5 } # define g()` | `g <- function(t) { return ((t^2+1)^0.5) } # define g()` | Print 3 |
| 295 | `returns the minimum value of d[i,j], i != j, and the row/col attaining`<br>`that minimum, for square symmetric matrix d; no special policy on`<br>`ties;`<br>`motivated by distance matrices` | `# returns the minimum value of d[i,j], i != j, and the row/col attaining`<br>`# that minimum, for square symmetric matrix d; no special policy on`<br>`# ties;`<br>`# motivated by distance matrices` | Print 3 |
| 345 | As of this writing, GPU has not yet become common among R users. | As of this writing, GPU **programming** has not yet become common among R users. | Print 3 |