**Online Appendix Chapter 21: The bivariate binormal model**

# Online Appendix 21.A: Sampling the bivariate normal distribution

The code below, in file “mainBivariateSampling.R”, illustrates sampling from the bivariate binormal model for the following parameter values:



 (mu12 in the code) denotes the mean of the distribution (of the z-samples) for diseased cases for modality 1, and  (mu22 in the code) denotes the mean of the distribution for diseased cases for modality 2; the corresponding non-diseased distributions have zero means.

The standard deviations of the non-diseased distributions are unity,  (sig1 in the code) is the standard deviation of the diseased distributions for modality 1,  (sig2 in the code) is the standard deviation of the diseased distributions for modality 2,  (rho1 in the code) is the *inter-modality correlation* of the z-samples for non-diseased cases and  (rho2 in the code) is the inter-modality correlation of the z-samples for diseased cases.

### Online Appendix 21.A.1: Code Listing

# mainBivariateSampling.R

rm(list = ls())

library(plotly)

library(MASS)

seed <- 1;set.seed(seed);N <- 10000 #number of samples

mu12 <- 1.5 # mean of diseased distribution, modality 1, truth 2

mu22 <- 2.0 # mean of diseased distribution, modality 2, truth 2

sig1 <- 1.0 # sq. root of variance of diseased distribution, modality 1

sig2 <- 1.5 # sq. root of variance of diseased distribution, modality 2

rho1 <- 0.3 # correlation non-diseased cases

rho2 <- 0.6 # correlation diseased cases

mu1 <- as.vector(array(0, dim = 2)) # vector, non diseased cases

mu2 <- as.vector(c(mu12, mu22)) # vector diseased cases

COV1 <- array(dim = c(2,2))

COV1[1,1] <- 1;COV1[2,2] <- 1;COV1[1,2] <- rho1;COV1[2,1] <- rho1

COV1 <- matrix(COV1, nrow = 2)

z1 <- mvrnorm(N, mu1, COV1)

mu1Est <- c(mean(z1[,1]), mean(z1[,2]))

rho1Est <- cor(z1[,1],z1[,2])

COV1Est <- cov(z1)

cat("expected means of non-diseased cases = ", 0, 0, "\n")

cat("expected correlation of non-diseased cases = ", rho1, "\n")

cat("expected covariance of non-diseased cases = \n")

print(COV1)

cat("\n")

cat("observed means of non-diseased cases = ", mu1Est, "\n")

cat("observed correlation of non-diseased cases = ", rho1Est, "\n")

cat("observed covariance of non-diseased cases = \n")

print(COV1Est)

cat("\n")

COV2 <- array(dim = c(2,2))

COV2[1,1] <- sig1^2;COV2[2,2] <- sig2^2;COV2[1,2] <-

rho2\*sig1\*sig2;COV2[2,1] <- rho2\*sig1\*sig2

COV2 <- matrix(COV2, nrow = 2)

z2 <- mvrnorm(N, mu2, COV2)

mu2Est <- c(mean(z2[,1]), mean(z2[,2]))

rho2Est <- cor(z2[,1],z2[,2])

COV2Est <- cov(z2)

cat("expected means of diseased cases = ", mu12, mu22, "\n")

cat("expected correlation of diseased cases = ", rho2, "\n")

cat("expected covariance of diseased cases = \n")

print(COV2)

cat("\n")

cat("observed means of diseased cases = ", mu2Est, "\n")

cat("observed correlation of diseased cases = ", rho2Est, "\n")

cat("observed covariance of diseased cases = \n")

print(COV2Est)

upperX <- ceiling(mu12 + 3 \* sig1)

upperY <- ceiling(mu22 + 3 \* sig2)

x <- seq(-3, upperX, by = 0.1)

y <- seq(-3, upperX, by = 0.1)

z <- array(dim = c(length(x), length(y)))

for (ix in 1:length(x)){

for (iy in 1:length(y)){

z[ix, iy] <- max(dmvnorm(c(x[ix], y[iy]), sigma = COV1),

dmvnorm(c(x[ix], y[iy]), mean = c(mu12, mu22), sigma = COV2))

}

}

p <- plot\_ly(x = x, y = y, z = z, type = "surface")

print(p)

Line 4 loads the MASS package, which contains functions related to the multivariate normal distribution, of which the bivariate normal distribution is a special case. The needed function, mvrnorm, is called at line 19[[1]](#footnote-1). Look at its Help page (enter mvrnorm in the Help window), Figure 1. Its first argument (N in the code) is the desired number of samples, initialized at line 5 to a large value. The second argument is the means vector, i.e., two values for bivariate sampling, . The third argument is the covariance matrix of the desired samples.

Lines 16 – 32 pertain to bivariate sampling from *non-diseased* cases. Line 16-18 initializes the covariance matrix (COV1 in the code) corresponding to non-diseased cases.

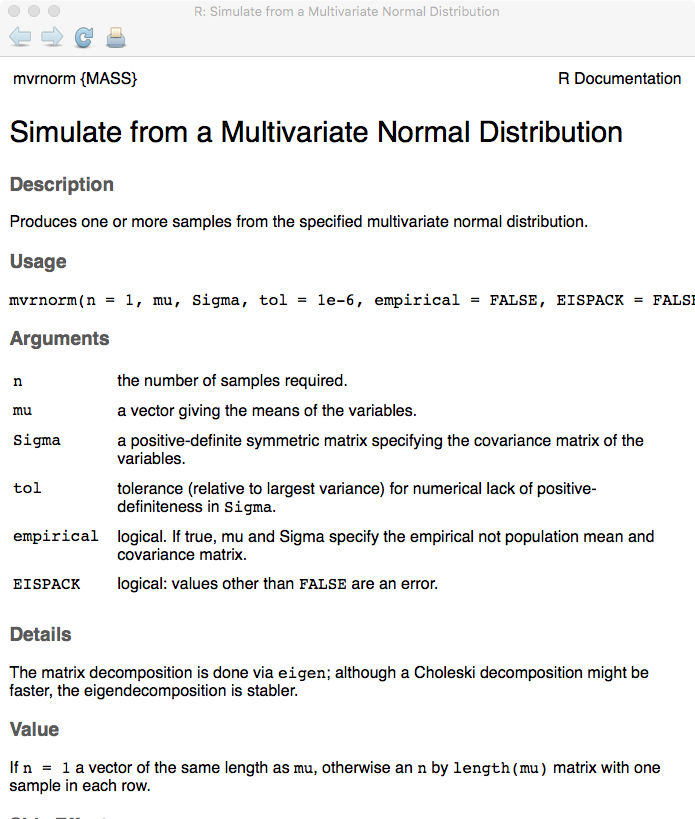


Figure : Help page for function mvrnorm called at line 19.

Insert a break point at line 34 and source the code. The relevant arguments are shown below:

Browse[2]> mu1

[1] 0 0

Browse[2]> COV1

[,1] [,2]

[1,] 1.0 0.3

[2,] 0.3 1.0

Line 19 samples the bivariate normal distribution for the non-diseased cases.

Browse[2]> str(z1)

num [1:10000, 1:2] -0.0292 0.7731 -0.0612 1.9875 0.5617 ...

- attr(\*, "dimnames")=List of 2

..$ : NULL

..$ : NULL0

In other words, z1 is a 10000 x 2 array. For each of the 10,000 non-diseased cases one has two z-samples, as in following example (the two values for the first 10 non-diseased cases are shown).

Browse[2]> z1[1:10,]

[,1] [,2]

[1,] -0.02921422 -0.98091220

[2,] 0.77310699 -0.47699102

[3,] -0.06115692 -1.28625373

[4,] 1.98754346 0.58476954

[5,] 0.56172167 -0.03040635

[6,] -0.35089525 -0.97207027

[7,] 0.57189962 0.21405612

[8,] 0.31603629 0.87447652

[9,] 0.61115569 0.31726384

[10,] -0.99123458 0.49881060

The 10,000 pairs of values are used to calculate estimates of the population values. For example, mu1Est is an estimate of mu1. Line 20 uses the function mean() to calculate the means of the two columns of z1, concatenates them and assigns the result to the variable mu1Est. Line 21 uses the function cor() to calculate the Pearson correlation between the two columns of z and assigns the result to rho1Est. Line 22 uses the cov() function to calculate the sampled covariance of z1, and the resulting 2x2 matrix is assigned to COV1Est. Lines 23 – 31 print the expected (i.e., true or population) values and the corresponding estimates of all model parameters.

> debugSource('~/onlinebookk21778/Ch21/software/mainBivariateSampling.R')

expected means of non-diseased cases = 0 0

expected correlation of non-diseased cases = 0.3

expected covariance of non-diseased cases =

[,1] [,2]

[1,] 1.0 0.3

[2,] 0.3 1.0

observed means of non-diseased cases = -0.002791453 -0.007749206

observed correlation of non-diseased cases = 0.3194812

observed covariance of non-diseased cases =

[,1] [,2]

[1,] 1.0051009 0.3225885

[2,] 0.3225885 1.0143726

Because of the large number of samples, the estimates are close to the expected values.

Line 34 – 50 repeats the process for 10,000 *diseased* cases. Exit debug mode, clear the break point, insert a new break point at line 52 and source the code.

> debugSource('~/onlinebookk21778/Ch21/software/mainBivariateSampling.R')

expected means of non-diseased cases = 0 0

expected correlation of non-diseased cases = 0.3

expected covariance of non-diseased cases =

[,1] [,2]

[1,] 1.0 0.3

[2,] 0.3 1.0

observed means of non-diseased cases = -0.002791453 -0.007749206

observed correlation of non-diseased cases = 0.3194812

observed covariance of non-diseased cases =

[,1] [,2]

[1,] 1.0051009 0.3225885

[2,] 0.3225885 1.0143726

expected means of diseased cases = 1.5 2

expected correlation of diseased cases = 0.6

expected covariance of diseased cases =

[,1] [,2]

[1,] 1.0 0.90

[2,] 0.9 2.25

observed means of diseased cases = 1.509351 2.009172

observed correlation of diseased cases = 0.6100278

observed covariance of diseased cases =

[,1] [,2]

[1,] 1.0247098 0.9295477

[2,] 0.9295477 2.2659128

Line 52 – 65 displays the 3D plot. The plot\_ly function (the needed package was loaded at line 3) produces an interactive plot. The initial view is shown in Figure 2.

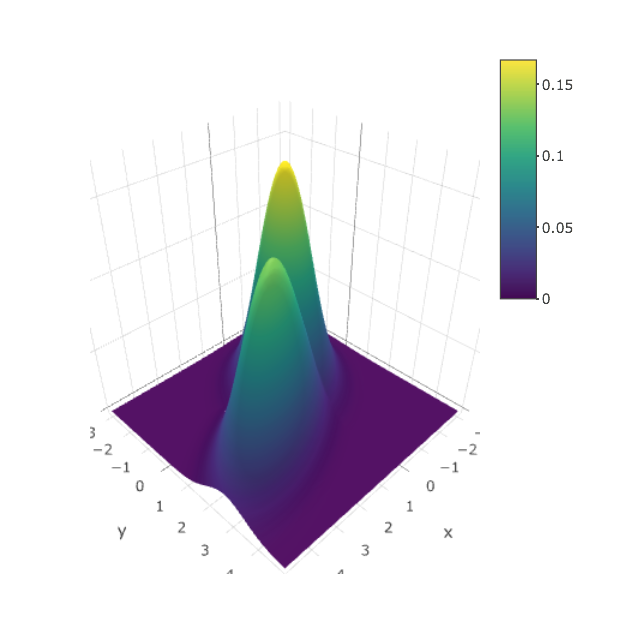


Figure : Result of sourcing the code. The interactive plot is created by plot-ly.

The plot, which appears in the plots window, may be too small to appreciate. Use the "*Show in new window*" icon, next to the broom symbol, to maker it larger. The reader is encouraged to experiment with the capabilities of the interactive plotting function; it is a good way to understand the bivariate distribution. Click on the figure, upon which new icons appear in the plots window (to the top right of the plot area); hover over each icon to reveals what it does; select the "*turntable rotation*" icon and rotate the plot (click and drag). The plot can be rotated much as a 3D object. One example is shown in Figure 3. Positioning (without clicking) the cursor over the plot shows the x, y and z values.

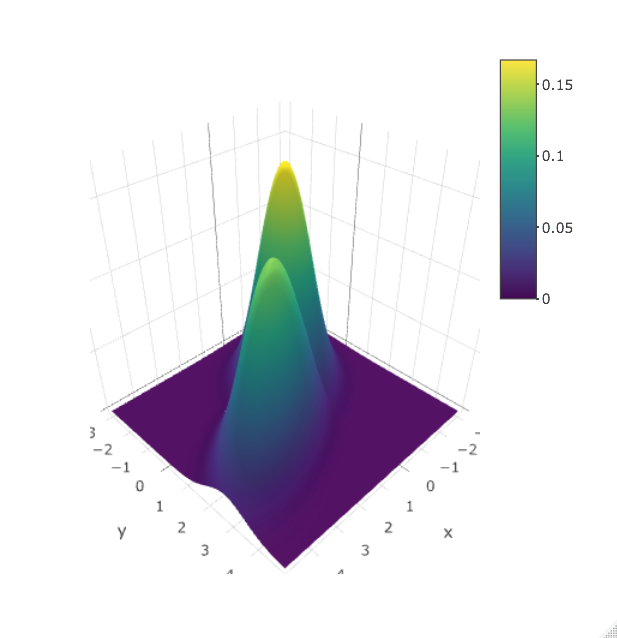


Figure : Plot window after rotation.

The reader may wish to explore mvrnorm in the Help menu in the lower-right window of RStudio to discover the other functions provided by the R package MASS (MASS: Support Functions and Datasets for Venables and Ripley's MASS1. R is descended from S). Bivariate is a special case of multivariate; depending on the dimensions of the mean and covariance matrix, this function can provide samples from an arbitrary multivariate normal distribution.

### Online Appendix 21.A.2

This section is intentionally blank.

# Online Appendix 21.B: Density of multivariate normal distribution

This section explores the bivariate density function (used in the preceding example) in more detail. The package is called mvtnorm, which may need to be installed, as it is not part of the standard R installation:

### Online Appendix 21.B.1 Code Snippet

> source(...)

Error in library(mvtnorm) : there is no package called ‘mvtnorm’

Installing a package has already been described, but a little repetition does not hurt. In the lower-right RStudio window one sees a tab called Packages, clicking on which reveals an icon called Install, and clicking on this opens a window called Install packages, the 2nd line allows one to enter the name of the package, and by the time one enters mvt it gives three choices, Figure 4, one of which is mvtnorm, selection of which and clicking on Install installs the package.

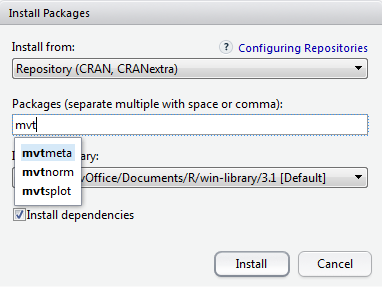


Figure : Installing a package under RStudio.

One may see a series of gibberish with the satisfying final line that states, in so many words, that the package has been downloaded.

The code below illustrates evaluation of the bivariate *pdf* for the following parameter values:



Open the file mainBivariateDensity.R. The listing follows:

### Online Appendix 21.B.2 Code Listing

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

library(mvtnorm)

X <- c(0.1,0.2)

mu1 <- 0;mu2 <- 0

rho <- 0.5;var1 <- 1.5;var2 <- 1.5;

COV <- matrix(c(var1, rho\*sqrt(var1\*var2), rho\*sqrt(var1\*var2), var2),2)

ret <- dmvnorm(X, sigma = COV)

cat("rho =", rho, ", var1 =", var1, ", var2 =", var2, "\n")

cat("density at 0.1, 0.2 = ", ret, "\n")

Source the code, to obtain the following output (one may get an ignorable warning that mvtnorm was built under an older version of R):

### Online Appendix 21.B.3 Code Output

> source(...)

rho = 0.5 , var1 = 1.5 , var2 = 1.5

density at 0.1, 0.2 = 0.1208948

# Online Appendix 21.C: Bivariate integrals

Bivariate integrals occur in the likelihood function. See book Equations 21.13 and 21.14. Open mainBivariateIntegrals.R.

### Online Appendix 21.C.1 Code Listing

# mainBivariateIntegrals.R

rm(list = ls())

library(mvtnorm)

cat("Integrals of the bivariate normal distribution\n")

mu <- c(0,0)

rho <- 0.5;var1 <- 1.5;var2 <- 1.5;

COV <- matrix(

c(var1, rho\*sqrt(var1\*var2),

rho\*sqrt(var1\*var2), var2),2)

ret <- pmvnorm(

c(-Inf, -Inf),

c(Inf, Inf),

mean=mu,

sigma = COV)

cat("Over the entire space = ", ret, "\n")

ret <- pmvnorm(

c(-Inf, -Inf),

c(Inf, 0),

mean=mu,

sigma = COV)

cat("Over the full space in one dimension\n")

cat("and the -ve half space in other dimension = ",

ret, "\n")

ret <- pmvnorm(

c(.3, .4),

c(.4, .5),

mean=mu,

sigma = COV)

cat("Between specified ctff. values = ",

ret, "\n")

Sourcing this code results in the following output:

### Online Appendix 21.C.2 Code Output

> source('~/book/BookCode/Chapter 7/mainBivariateIntegrals.R')

Integrals of the bivariate normal ...

Over the entire space = 1

Over the full space in one dimension and the -ve half space in the other dimension = 0.5

Between specified ctff. values = 0.001136472

Line 2 loads the needed package; line 5 prints a helpful message describing the output that is to follow. Lines 6-10 initialize the parameters of the models to the following values:



Line[[2]](#footnote-2) 12-17 calculates and prints the integral when the lower thresholds are at negative infinity in both conditions and the upper thresholds are at positive infinity, also in both conditions. Since we are including all possible values of the two random variables, this integral must be unity. The relevant function is pmvnorm, which is the analog of pnorm for the univariate case. The first argument is an array of the two lower limits; the second argument is an array of the two upper limits, the other arguments are the means vector and the covariance matrix.

ret <- pmvnorm(

c(-Inf, -Inf),

c(Inf, Inf),

mean=mu,

sigma = COV)

The first argument, occurring at line 13, is c(-Inf,-Inf). The second argument, occurring at line 14, is c(Inf, Inf). As expected, the result is unity.

Line 18-25 calculates and prints the integral when the lower thresholds are at negative infinity in both conditions and the upper thresholds are at positive infinity in the 1st condition but zero in the 2nd condition. Since we are including all possible values of the 1st random variable but only half the possible values of the 2nd random variable, this integral must be 0.5. Line 26 - 32 does a more general case: it calculates the integral under the bivariate normal between thresholds 0.3 and 0.4 in the 1st condition and between thresholds .4 and .5 in the 2nd condition. In other words,



Compare this to the symbolic form shown in book Eqn. 21.14.

# Online Appendix 21.D: Running CORROC2 in a Mac environment

[Windows users can skip this section]

The procedure for running CORROC2, a Windows program, on a MAC requires installation of a virtual machine and installing Windows on it. One option is VMware Fusion (the author has Version 10[[3]](#footnote-3)); it is not free but is affordable (around $80) and the software runs on almost all operating systems. Next, Windows 8 was installed under VMWare (a Windows CD-ROM installation disk was needed). Figure 5 is a typical starting screen when VMware is started on a MAC and one has "logged in" to Windows 8.

|  |
| --- |
|  |

Figure : This screen is intended to guide one through the process of setting up the server that runs on the Windows side and with which the OS X side communicates with the Windows side. This is what the VMware Fusion screen looks like when running Windows 8 on an OS X computer.

Next one needs to download *secure shell server software*; this allows the MAC side to communicate with the Windows side. One option is freeSSHd, which needs to be downloaded under Windows[[4]](#footnote-4). One needs to start the software in Administrator mode (right click on the icon, the blue partially filled box with a lock symbol in the bottom ribbon visible in Figure 5) and create user dev (could be case-sensitive) with password 1234 (the aim being to communicate between the Mac side and the Windows side and assuming one's Mac is secure, additional security is probably not needed, but the author does not guarantee this, lest he be sued). Figure 6 shows the relevant freeSSHd screens. Click on Add to add user dev, or your preferred name, and carefully follow the guidance provided by Figure 6.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Figure : This screen is intended to guide one through the process of setting up the server that runs on the Windows side and with which the OS X side communicates. This screen shows the process of setting up the secure shell server software, freeSSHd.

Starting from Figure 5, run ipconfig under the Windows command line (Start Menu 🡪 Run 🡪 cmd) to get the IP4 address of Windows, Figure 7; in the author's case it was 172.16.36.80 (the last number can change from time to time, which gave the author headaches).

**Note added Dec 31, 2017**

The following link describes how to set up a static IP4 address: <https://willwarren.com/2015/04/02/set-static-ip-address-in-vmware-fusion-7/>. To determine your virtual Windows 8 computer's name (which is needed in the directions provided in the cited website) use ⌘-Ctrl-c in Windows 8, which brings up a narrow vertical menu bar at the far right. Select Settings and one sees the computer name, which in the author's computer was WIN-4K8HTNR0NKB.

Using these directions the author set up his virtual machine so that the IP4 address is always 172.16.36.80. In the interest of time the author has not updated all figures containing this address. Just be advised that the number in question actually ends with 80.

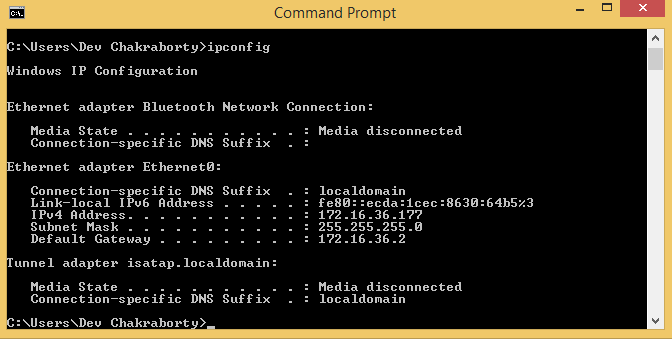


Figure : This screen shows the process of getting the IP4 address, 172.16.36.177 in the current example. Your value may be different, and may change between sessions. Enter the correct value at line 2 of CORROC2.sh and save the file. The address actually ends with .80.

A "script" file named CORROC2.sh, reproduced below, handles the communication between the two operating systems. Lines 2 - 5 have the login information that one set using freeSSHd. One may need to modify line 2 based on the IP4 address found above.

### Online Appendix 21.D.1 Code Listing

#!/usr/bin/expect –

spawn ssh -l dev 172.16.36.80

expect "password:"

send "1234\r"

expect ">"

send "net use X: /delete\r"

expect ">"

send "net use X: \"\\\\vmware-host\\Shared Folders\\Debug 2\"\r"

expect ">"

send "cd /D X:\r"

expect ">"

send "CORROC2.bat\r"

expect ">"

send "cd /D C:\r"

expect ">"

send "exit\r"

expect eof

OSX communicates with virtual Windows 8 using the secure shell (SSH) service. To start the service, open the Terminal window on the Mac and carefully type in everything on this line, as in Figure 8, ensuring that the IP4 address is the correct one, and hit return.

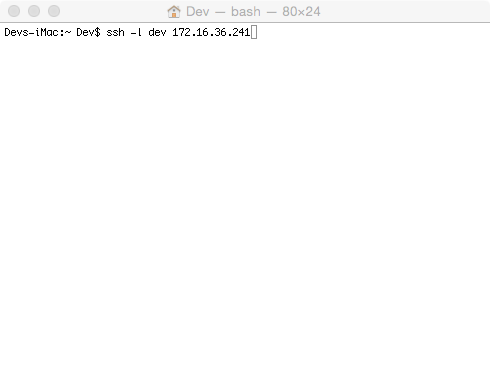


Figure : Running the Terminal window on the Mac side and initiating the connection to the Windows side. The address actually ends with .80.

One may get a message from OSX to the effect "*are you sure (yes/no)"* to which one needs to respond "*yes*" and next one will be asked the password, which is "1234" (enter the numbers, not the quotation marks). The last line in the Terminal window should say C:\WINDOWS\system32>, Figure 9. If so, the MAC side is communicating with the Windows side and one need not repeat this process until a new session is started.

**Do not close this window until you are done with the following demonstrations. Closing it will terminate the secure shell service.**

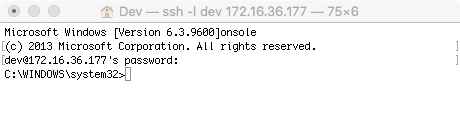


Figure : The connection worked! The password, 1234, of course, is not shown. The address actually ends with .80. **Do not close this window.**

The folder ~\onlinebookk21778\Ch21\software\corrocii\Debug (case sensitive!) needs to be shared with VMware. The steps are shown below. In VMWare, find Virtual Machine Sharing Sharing Settings, Figure 10.

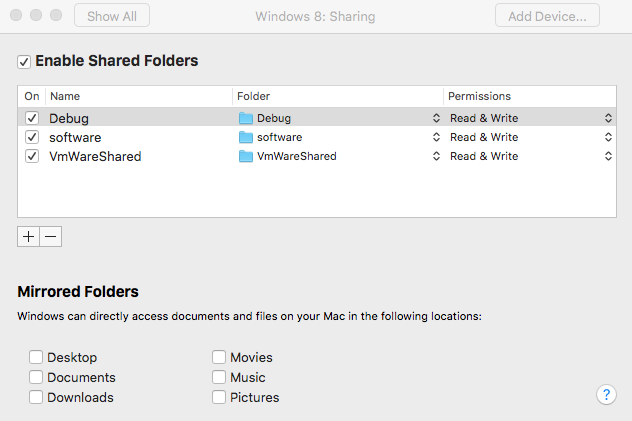


Figure : Sharing the folder containing the Windows CORROC2 software under VMWare. Delete the existing shared Debug folder (- icon) and use the + icon to add your folder; use the Finder to run down the directory tree, starting from onlinebookk21778.

Test the connection by clicking on Run Script, Figure 11 (again, make sure one is using the current IP4 address).

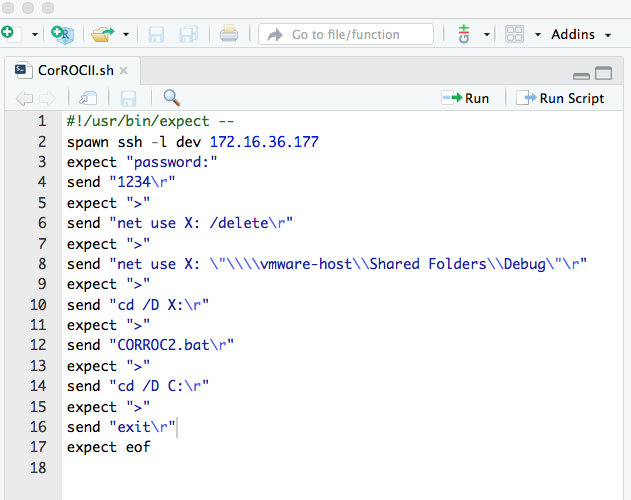


Figure : Testing the connection between the Mac and Windows. The address at line 2 actually ends with .80.

One should see the following message:

> system("./CORROC2.sh")

> system("./CorROCII.sh")

spawn ssh -l dev 172.16.36.80

dev@172.16.36.177's password:

*The absence of error messages signals success*!

As a further check, under RStudio, in the File panel, click on the folders corrocii, Debug, in that order, as shown in Figure 12, to find the folder that contains the CORROC2 files (the middle panel in Figure 12 shows the original FORTRAN files that the author downloaded decades ago, and held on to, which made all this possible; the current University of Chicago ROC website does not provide these files):

|  |  |  |
| --- | --- | --- |
|  |  |  |

Figure : Finding the folder containing the CORROC2 files. Starting from the original Files panel, left figure, click on corrocii icon to get to the middle figure, then click on the Debug icon to get to the right figure. Notice the time stamp on file CorrocIIOutput.txt.

The date and time stamp on the file CorrocIIOutput.txt should reflect a recently overwritten file (the reader's time). Click on the file – it contains the output of running CORROC2.exe as explained in the book. Alternatively, one can delete this file (under RStudio, check the box next to the file name, then click Delete) and Run Script again, Figure 11, to recreate it.

# Online Appendix 21.E: Application to a real dataset

This description augments book section 21.7. It demonstrates application of CORROC2 to a real dataset (in VanDyke.lrc). The file in question is mainCorroc2.R, whose listing follows:

## Online Appendix 21.E.1 Code Listing

rm(list = ls()) # mainCorroc2.R

library(RJafroc)

source("RunCorrocOnPairedData.R")

jSelected <- 1

dataFile <- "VanDyke.lrc"

dataset <- DfReadDataFile(dataFile, format = "MRMC")

dataset <- DfExtractDataset(dataset, rdrs=jSelected)

K1 <- dim(dataset$NL)[3]-dim(dataset$LL)[3]

K2 <- dim(dataset$LL)[3]

z1b <- dataset$NL[,1,1:K1,1]

z2b <- dataset$LL[,1,1:K2,1]

retCorrocii <- RunCorrocOnPairedData (z1b,z2b,desiredNumBins)

if (length(retCorrocii) == 1) stop("CorrocII failed") else # skip degenerate readers

{

parametersOrig <- retCorrocii$parms

Cov <- retCorrocii$Cov

cat("The 6 parameters are ", parametersOrig, "\n")

cat("The 2 sided pValue is ", retCorrocii$pValue, "\n")

cat("The covariance matrix is:\n")

print(Cov)

}

## Online Appendix 21.E.2 Explanation of the code

Line 2 loads the RJafroc package. Line 3 sources the file RunCorrocOnPairedData.R, which does exactly what its name suggests – it converts two-modality single-reader data to the format shown in the section on CORROC2 data input and calls the CORROC2 Windows executable file. Since the dataset contains multiple reader data in two modalities, line 5 sets jSelected (for selected reader) to 1; i.e., data for the 1st reader in both modalities will be analyzed by CORROC2. A limitation of CORROC2, not present in ROC-kit, is that it can only analyze binned data. One ROC data file VanDyke.lrc is included with the software. Line 7 uses the function DfReadDataFile() to read the contents of the Van Dyke dataset.

Line 8 uses DfExtractDataset() to extract data for the selected reader, reader 1 in the current example. The next two lines determine the numbers of non-diseased and diseased cases, respectively. Lines 11 and 12 create two arrays, z1b and z2b, corresponding to the ratings in the two modalities for non-diseased and diseased cases, respectively. Line 13 passes these ratings to the function RunCorrocOnPairedData(). This function needs to know the number of bins, 5 for the Van Dyke dataset, as it determines the length of the covariance matrix output. Line 14 checks if CORROC2 failed to converge as MLE algorithms sometimes do not converge. If the program did converge, the output values are displayed. Source the file mainCorroc2.R; one should see the output listed in book section 21.7.1.

# References

1. Venables WN, Ripley BD. *Modern applied statistics with S-PLUS.* Springer Science & Business Media; 2013.

2. Hillis SL. A comparison of denominator degrees of freedom methods for multiple observer ROC studies. *Statistics in Medicine.* 2007;26:596-619.

3. Hillis SL, Berbaum KS. Power Estimation for the Dorfman-Berbaum-Metz Method. *Acad Radiol.* 2004;11(11):1260-1273.

4. Hillis SL, Berbaum KS. Monte Carlo validation of the Dorfman-Berbaum-Metz method using normalized pseudovalues and less data-based model simplification. *Acad Radiol.* 2005;12(12):1534-1541.

5. Hillis SL, Berbaum KS, Metz CE. Recent developments in the Dorfman-Berbaum-Metz procedure for multireader ROC study analysis. *Acad Radiol.* 2008;15(5):647-661.

6. Hillis SL, Obuchowski NA, Berbaum KS. Power Estimation for Multireader ROC Methods: An Updated and Unified Approach. *Academic Radiology.* 2011;18(2):129-142.

7. Hillis SL, Obuchowski NA, Schartz KM, Berbaum KS. A comparison of the Dorfman-Berbaum-Metz and Obuchowski-Rockette methods for receiver operating characteristic (ROC) data. *Statistics in Medicine.* 2005;24(10):1579-1607.

1. The mvtnorm package, also loaded at line 4, is needed for the function dmvnorm, i.e., the multivariate normal *pdf* or density function, called at line 59 – 60. [↑](#footnote-ref-1)
2. In case the reader is wondering why split the code over so many lines, during the proofing process the author received the following comment many times from the publisher: "*AU: Please check and advise on the code breaking throughout the book and also check and verify the line numbers for accuracy*". Due to page width limitation, a long line of code would "break" over one or more lines. Rather than explain that R does not care if the code "breaks" over lines, the author chose to keep the individual lines short and avoid "code breaking". [↑](#footnote-ref-2)
3. With future OSX upgrades it may be necessary to also upgrade VMware (a "forced" upgrade). [↑](#footnote-ref-3)
4. http://www.freesshd.com/?ctt=overview [↑](#footnote-ref-4)