Principal Component Analysis in R

An examination of the different functions and methods to perform PCA

Gregory B. Anderson

Introduction

Principal component analysis (PCA) is a multivariate procedure aimed at reducing the dimensionality of multivariate data while accounting for as much of the variation in the original data set as possible. This technique is especially useful when the variables within the data set are highly correlated and when there is a higher than normal ratio of explanatory variables to the number of observation. Principal components seeks to transform the original variable to a new set of variables that are (1) linear combinations of the variables in the data set, (2) uncorrelated with each other, and (3) ordered according to the amount of variation of the original variables that they explain (Everitt and Hothorn 2011).

In R there are two general methods to perform PCA without any missing values: (1) spectral decomposition (R-mode [also known as eigendecomposition]) and (2) singular value decomposition (Q-mode; R Development Core Team 2011). Both of these methods can be performed longhand using the functions `eigen` (R-mode) and `svd` (Q-mode), respectively, or can be performed using the many PCA functions found in the stats package and other additional available packages. The spectral decomposition method of analysis examines the covariances and correlations between variables, whereas the singular value decomposition method looks at the covariances and correlations among the samples. While both methods can easily be performed within R, the singular value decomposition method (i.e., Q-mode) is the preferred analysis for numerical accuracy (R Development Core Team 2011).

This document focuses on comparing the different methods to perform PCA in R and provides appropriate visualization techniques to examine normality within the statistical package. More specifically this document compares six different functions either created for or can be used for PCA: `eigen`, `princomp`, `svd`, `prcomp`, `PCA`, and `pca`. Throughout the document the essential R code to perform these functions is embedded within the text using the font Courier New and is color coded using the technique provided in Tinn-R (https://sourceforge.net/projects/tinn-r). Additionally, the results from the functions are compared using simulation procedure to see if the different methods differ in the eigenvalues, eigenvectors, and scores provided from the output.

Examining Normality

Although principal component analysis assumes multivariate normality, this is not a very strict assumption, especially when the procedure is used for data reduction or exploratory purposes. Undoubtedly, the correlation and covariance matrices are better measures of similarity if the data is normal, and yet, PCA is often unaffected by mild violations. However, if the new components are to be used in further analyses, such as regression analysis, normality of the data might be more important.

R provides a useful graphical interface to explore the assumptions of normality. From a univariate perspective, one can assess normality by looking at normal probability plots where the data is plotted against a theoretical normal distribution (Montgomery et al.2006). If the variable is normally distributed it should approximate a straight line. In R this can be visualized using the functions `qqnorm` and `qqline` from the base package (R Development Core Team. 2011):

```
qqnorm(X[,1]);qqline(X[,1])
```

where `X[,1]` is a vector of observations for a variable in a dataset. For a more sophisticated plot including confidence bands, a similar command can be called from the car package (Fox and Weisberg 2011), and one can extend this command to plot all variables in the dataset in one graphing window:

```
par(mfrow=c(1,3));for(i in 1:3){qqPlot(X[i],ylab=colnames(X)[i])}
```

where the function `par` specifies the attributes of the graphing window (i.e., one row and three columns of plots), the `for` function initiates a loop over the three variables and `i` indexes the variables. To asses normality for bivariate data we can examine a scatterplot matrix of the variables of the dataset. This plot is fairly easy to produce in R; however, the arguments can be easily extended to customize the graph (e.g., adding histograms on the diagonal [see Appendix A and B for an example]). The call for this graph is:

```
pairs(X)
```

Finally, because many of the ordinational procedures assume multivariate normally distributed data, it is important to ensure that the data reflect this pattern. One can assess for this in `R` by producing a kernel density estimate plot and a multivariate normality plot using the following functions and arguments:

```
mu<-colMeans(X);sig<-cov(X)
stopifnot(mahalanobis(X,0,diag(ncol(X)))==rowSums(X*X))
D2<-mahalanobis(X,mu,sig);n=dim(X)[1];p=dim(X)[2]
par(mfrow=c(2,1))
plot(density(D2,bw=.5),main=paste("Mahalanobis Distances))
qqplot(qchisq(ppoints(n),df=p),D2,main="Q-Q Plot of Mahalanobis D Squared vs.
Quantiles of Chi Squared")
abline(0,1,col='gray')
```

where the kernel density plot should look like a normal curve and the qqplot should resemble a straight line (see Appendix B.b for the resulting graph). All of these plotting functions can be combined into a single function that acts as a wrapper to produce these graphs in multiple plotting windows. This function can be found in Appendix A, where the call is simplified to:

```
exam.norm(X,univariate=T,univariate.test=F,bivariate=T,multivariate=T)
```

The `univariate.test` argument performs the Shapiro-Wilk test of normality available in the stats package (R Development Core Team. 2011) for each of the variables in the dataset. Output from this function can be found in Appendix B.

PCA USING SPECTRAL DECOMPOSITION IN R

ANALYSIS USING THE R FUNCTION `eigen`

The function `eigen` computes eigenvalues and eignvectors using spectral decomposition of real or complex matrices. The arguments for the function are as follows: `X` (a matrix to be used [for PCA we specify either the correlation matrix or the covariance matrix]), `symmetric` (a logical argument; if `TRUE` the matrix is assumed to be symmetric [if not specified, the matrix will be inspected for symmetry]), and `only.values` (a logical argument; if `TRUE` only the eigenvalues are computed and returned). The function is specified as follows:

```
e.cor<-eigen(cor(X))#For the correlation matrix (i.e., scaled and centered)
e.cov<-eigen(cov(X))#For the covariance matrix
```

The object that is created (i.e., either `e.cor` or `e.cov`) is a list with two components: values and vectors. Values is a vector containing the `p` eigenvalues of the data matrix, ordered by the amount of variation that is explained. Vectors is a `p` x `p` matrix containing the eigenvectors. The two components can be stored in new objects as follows:

```
eigenvalues<-e.cor$values #or eigenvalues<-e.cov$values for the unscaled data
eigenvectors<-e.cor$vectors
```

Although the `eigen` function only provides the eigenvalues and eigenvectors within the created object, the new variables (scores), proportion of variance explained, cumulative variance explained, and the standard deviation of the new variables can be calculated from these two new objects. For sake of brevity, only the script for the scaled data (i.e., using the correlation matrix) is shown below:

```
X2=scale(X) #New scaled dataset with mean of 0 and sd of 1
scores<-X2%*%eigenvectors #New Variables
total.var<-sum(diag(cov(X2))) #Calculate total variance in scaled data
prop.var<-rep(NA,ncol(X));cum.var<-rep(NA,ncol(X)) #Create empty vectors
#Calculate proportion of variance explained and cumulative variance explained
for(i in 1:ncol(X)){prop.var[i]<-var(scores[,i])/total.var}
for(i in 1:ncol(X)){cum.var[i]<-sum(prop.var[1,i])
sdev=sqrt(eigenvectors) #Std of the new components
```

For ease of use, this script can be formulated in a wrapper function to calculate these values automatically. This function can be found in Appendix C, and to perform PCA with this function the call is as follows:

```
pc_data1<-PC(X,method="eigen",scaled=T,graph=F,rm.na=T,print.results=T)
```

TABLE 1. Outputs of different functions to do principal component analysis using spectral decomposition in R. The PC function is a wrapper created to calculate eigenvalues, eigenvectors, scores, standard deviations, and the variance explained (see Appendix C for function script).

eigen(cor(X))	PC(X, method="eigen")	princomp(X)	Interpretation
values	values	sdev*sdev	Eigenvalues
vectors	vectors	loadings	Eigenvectors
	scores	scores	Scores
sqrt(values)	sdev	sdev	Standard deviations of each column of the rotated data
	colMeans(X)	center	Mean value used for centering
	importance[2,]		Prop. Var. Explained
	importance[3,]		Cum Var. Explained

The resulting object contains the vector of eigenvalues, the matrix of eigenvectors, the vector of standard deviations, a table of the variance explained by each new component, and the new components. These values can be extracted using the functions found in Table 1.

ANALYSIS USING THE R FUNCTION `princomp`

Like the function `eigen`, `princomp` performs principal component analysis using the spectral decomposition of a matrix (i.e., R-mode PCA). The calculation is actually done using `eigen` on either the correlation or covariance matrix; however, the function is called using either a data matrix or a formula with no response variable. The arguments for the function are as follows: `formula` (e.g., `~X1+X2`), `data` (optional data frame containing the variables), `subset` (an optional vector used to select particular rows of the data matrix), `na.action` (a function to indicate how missing values should be treated), `x` (a matrix or dataframe to be used), `cor` (a logical statement; if `TRUE` then the PCA is performed on the correlation matrix, if `FALSE` the PCA is performed on the covariance matrix), `scores` (a logical statement indicating if the scores should be calculated), `covmat` (a covariance matrix to be used instead of the covariance matrix of `x`), and `newdata` (an optional dataset to predict into). The basic arguments of the function are specified as follows:

```
pc1.cor<-princomp(X, cor=TRUE) #PCA performed with correlation matrix
pc1.cov<-princomp(X, cor=FALSE) #PCA performed with the covariance matrix
```

The object that is created (i.e., either `pc1.cor` or `pc1.cov`) is a list with seven components: `sdev` (the standard deviations of the principal components [equal to the square root of the eigenvalues]), `loadings` (a matrix containing the eigenvectors), `center` (the column means of the original data used to center each variable), `scale` (the original standard deviation of each column used to scale each variable), `n.obs` (the number of observations), `scores` (the new variables), and `call` (the call used to perform the analysis). These values can be extracted using the functions found in Table 1.

PCA USING SINGULAR VALUE DECOMPOSITION IN R

ANALYSIS USING THE R FUNCTION svd

The most general way of performing singular value decomposition in R is to use the function svd. The required arguments for the function only include the specification of a data matrix; however, all scaling and centering must be performed before running the analysis. To reduce the amount of output, the argument nu=0 can be added to the command, specifying not to compute any left singular vectors (not needed for PCA). The following script computes the singular value decomposition for scaled (var=1) and centered (mean=0) data.

```
scale.svd<-svd(scale(X),nu=0)
```

This command provides an object with two components: d (the singular values of the data) and v (the eigenvectors). From these components, one can calculate the eigenvalues, standard deviation of the new variables, proportion of variance explained, and the cumulative variance explained.

```
var<-nrow(X)-1   #Calculate the divisor for the variance
vectors<-scale.svd$v  #Specify eigenvectors for svd
sdev<-scale.svd$d/sqrt(var) #Calculate standard deviation of new variables
values<-sdev*sdev   #Sqaure the standard deviation to find the eigenvalues
scores<-scale(X)%*%vectors #Calculate the new components
total.var<-sum(diag(cov(scale(X)))) #Calculate total variance in scaled data
#Calculate proportion of variance explained and cumulative variance explained
prop.var<-rep(NA,ncol(X));cum.var<-rep(NA,ncol(X))
for(i in 1:ncol(X)){prop.var[i]<-var(scores[,i])/total.var}
for(i in 1:ncol(X)){cum.var[i]<-sum(prop.var[1:i])}
```

Similar to the analysis using the eigen function, one can create a wrapper function to calculate these values automatically. This function can be found in Appendix C. To perform PCA with this function the call is as follows:

```
pc_data2<-PC(X,method="svd",scaled=T,graph=F,rm.na=T,print.results=T)
```

The resulting object contains the vector of eigenvalues, the matrix of eigenvectors, the vector of standard deviations, a table of the variance explained by each new component, and the new components. These values can be extracted using the functions found in Table 2.

ANALYSIS USING THE R FUNCTION prcomp

The function prcomp performs a principal component analysis using singular value decomposition of the data matrix. Unlike the svd function, the data matrix does not need to be centered or scaled prior to analysis because these options can be specified as arguments in the prcomp command. The arguments of the function include: formula (e.g., ~X1+X2), data (optional data frame containing the variables), subset (an optional vector used to select particular rows of the data matrix), na.action (a function to indicate how missing values should be treated), x (a matrix or dataframe to be used), retx (a logical statement to determine if the rotated variables should be returned), center (a logical statement to indicate that the data should be centered at zero), scale (a logical statement to indicate that the data should be scaled [default=FALSE]), tol (an argument to indicate that the magnitude below the provided value for components should be omitted), and newdata (an optional dataset to predict into). The basic arguments of the function are specified as follows:

```
pc2.scale<-prcomp(X,scale=T,tol=0)  # Compare to princomp(X,cor=T)
pc2.center<-prcomp(X,scale=F,tol=0)  # Compare to princomp(X,cor=F)
```

The object that is created is a list with seven components: sdev (the standard deviations of the principal components [equal to the squared eigenvalues]), rotation (a matrix containing the eigenvectors), x (the new variables), center (the column means of the original data used to center each variable), and scale (the original variance of each column used to scale each variable). These values can be extracted using the functions found in Table 2.

ANALYSIS USING THE R FUNCTION PCA

The function PCA is found in the FactoMineR package (Husson et al. 2012), which provides several tools for multivariate exploratory data analysis. The function provides many additional options, including new plotting functions and a way to weight variables or observations. The basic arguments for the function are as follows: X (data frame or matrix of variables), ncp (the number of components to be kept [default=5], scale.unit (a logical statement specifying that the data should be scaled to a unit variance), graph (a logical statement specifying that a graph should be displayed), and axes (a vector specifying which components to plot). The basic command is:

```
pc3.scale<-PCA(X,ncp=ncol(X),scale.unit=T,graph=F)
pc3.center<-PCA(X,ncp=ncol(X),scale.unit=F,graph=F)
```

The object that is created contains a plethora of information found in many different lists and matrices. The matrix eig contains the eigenvalues (eig$eigenvalue), the proportion of variance explained by each component (eig[,2]) and the cumulative variance explained by each successive component (eig[,3]). The svd list provides the results from the singular value decomposition, including standard deviations (svd$vs) and eigenvectors (svd$V). The ind list provides the new variables (ind$coord) among other elements.

ANALYSIS USING THE R FUNCTION pca

The final function discussed in this document is the function pca from the bioconductor pcaMethods package (Stacklies et al. 2007). The pca function is actually a function wrapper for the svdPca function (and many others) which is in itself a wrapper for the prcomp function discussed above. The benefit of this function is that it provides different visualization techniques and a multitude of different analysis approaches to handle missing values, including: iterative methods (i.e, nipals or mipals), Bayesian methods (i.e., bpca), probabilistic models (i.e., ppca), and expectation maximation (i.e., svdImpute). The basic singular value decomposition method can be called with the following arguments: object (data frame or matrix of variables), method (for singular value decomposition one must specify svd), scale (statement to determine whether to scale to one unit variance [uv], by a vector [vector], by the square root unit variance [pareto] or to not scale at all [none]), center (logical statement to

TABLE 2. Outputs of different functions to do principal component analysis using singular value decomposition in R. The PCA function is part of the FactoMineR package (Husson et al. 2012) and the pca function is part of the bioconductor pcaMethods package (Stacklies et al. 2007). The PC function is a wrapper created to calculate eigenvalues, eigenvectors, scores, standard deviations, and the variance explained (see Appendix C for function script).

svd(scale(X))	**PC(X, method="svd")**	**prcomp(X)**	**PCA(X)**	**pca(X, method="svd")**	**Interpretation**
	values	sdev*sdev	eig$eigenvalue	sDev*sDev	Eigenvalues
v	vectors	rotation	svd$V	loadings	Eigenvectors
	scores	x	ind$coord	scores	Scores
	sdev	sdev	sqrt(eig$eigenvalue)	sDev	Standard deviations of each column of the rotated data
	colMeans(X)	center		center	Mean value used for centering
	importance[2,]	summary(X)$ importance[3,]	eig[,2]	R2	Prop. Var. Explained
	importance[3,]	summary(X)$ importance[3,]	eig[,3]	R2cum	Cum Var. Explained

*Note: to extract attributes from the pcaMethods function pca a @ symbol is used instead of a standard $ (e.g., pc@loadings, where pc is an object created to store the results of the pca function).

determine if the data should be centered at zero [or a vector of means can be provided]), and nPcs (the number of components to be kept). The command is performed as:

```
pc4.scale<-pca(X,method="svd",scale="uv",nPcs=ncol(X))
pc4.center<-pca(X,method="svd",scale="none",nPcs=ncol(X))
```

The returned object from the function contains the standard information, including: standard deviations (sDev), eigenvalues (sDev*sDev), eigenvectors (loadings), scores (scores), mean values (center), and the proportion and cumulative variance explained (R2 and R2cum, respectively). However, all of these components are extracted with the @ symbol rather than the $ sign as with most R objects.

SIMULATION STUDY TO COMPARE OUTPUT

Although similar output can be extracted from any of the above approaches to principal component analysis, the methods of calculating the eigenvalues and eigenvector can be very different (i.e., spectral vs. singular value decomposition). Additionally, each function might differ in the rounding it uses for values or elements with many decimal places. Therefore, to test if differences exist between the outputs, a simulation procedure was created to estimate the mean overall difference in eigenvectors and eigenvalues and consequently the new variables created (i.e., scores) for 1000 simulated datasets.

Simulated data sets were created using the function rmvnorm from the mvtnorm package (Genz and Bretz 2009; Genz et al. 2012) using a randomly generated correlation matrix (created using the rcorrmatrix function from the package clusterGeneration [Oiu and Joe 2009]) and mean values from randomly drawn numbers from a normal distribution. Datasets were created with five variables and 1000 observations and were randomly generated 1000 times. For each randomly generated dataset (i.e., one simulation), PCA was performed using five different functions: eigen (as called by the wrapper developed in Appendix C [i.e., PC]), princomp, prcomp, PCA, and pca. From each respective object created, the eigenvalues, eigenvectors, and scores were saved to new arrays where the first dimension represented the different functions, and the second (rows or observations) and third dimensions (columns or variables) contained the different elements (however, the eigenvalues only required a two dimensional array). To determine if differences in the values provided existed, the absolute value of each element from the different functions was subtracted from the mean absolute value of the element of the different functions used (e.g., eigen, princomp, etc.). The mean of these differences was then taken to determine the mean deviation observed for the particular output (i.e., eigenvalue, eigenvector, or score). Finally, the mean difference was averaged across all of the simulations (n=1000). If only minor differences occurred between the different functions, one would expect these values to be very close to zero. The script to perform this analysis can be found in Appendix D. Although this approach gives an overall assessment of the differences that exist between the different functions, it provides no way of determining if spectral and singular values methods give different answers. To test if such a scenario exists, the simulation procedure was repeated but this time only with two functions: `eigen` as performed by the wrapper function PC (Appendix C) and `svd` as performed by the wrapper function PC. The results from this simulation procedure give the average differences between the two functions.

The results from the analysis suggest that no major overall difference exist between the different functions. The mean values for the differences in eigenvalues, eigenvectors and scores when all of the functions were included were essentially zero (eigenvalues=-7.893972e-19, eigenvectors=3.977917e-20, and scores=1.526186e-20). Likewise the paired procedure of `eigen` and `svd`, found no substantial differences in the results (eigenvalues=-5.785100e-19, eigenvectors=-9.761018e-20, and scores=-1.940833e-21).

CONCLUSION

The R statistical package provides many different methods to perform PCA; however, the components of the output vary from package to package (e.g., some provide standard deviations rather than eigenvalues). Additionally, some packages provide advanced features such as the ability to weight values and ways to deal with missing values. Importantly, the results from these different functions for standard PCA do not differ markedly. This is somewhat surprising given that some functions use spectral decomposition while others use singular value decomposition. The differences in results, however, might become more substantial as the number of observations decreases and the number of variables increases.

REFERENCES

Everitt, B. and T Hothorn. 2011. An Introduction to Applied Multivariate Analysis with R (Use R!). Springer, New York, NY.

Fox, J. and S. Weisberg. 2011. An R Companion to Applied Regression. Thousand Oaks, CA, USA.

Genz, A. and F. Bretz. 2009. Computation of multivariate normal and t. probabilities. Lecture Notes in Statistics, Vol. 195., Springer-Verlage, Heidelberg.

Genz, A. F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, T. and Hothorn. 2012. mvtnorm: multivariate normal and t distributions. R package version 0.9-9992.

Husson, F. J.Josse, S. Le, and J. Mazet. 2012. FactoMineR. Multivariate exploratory data analysis and data mining with R. R package version 1.18.

Qiu, W. and H. Joe. 2009. clusterGeneration: random cluster generation (with specified degree of separation). R package version 1.2.7.

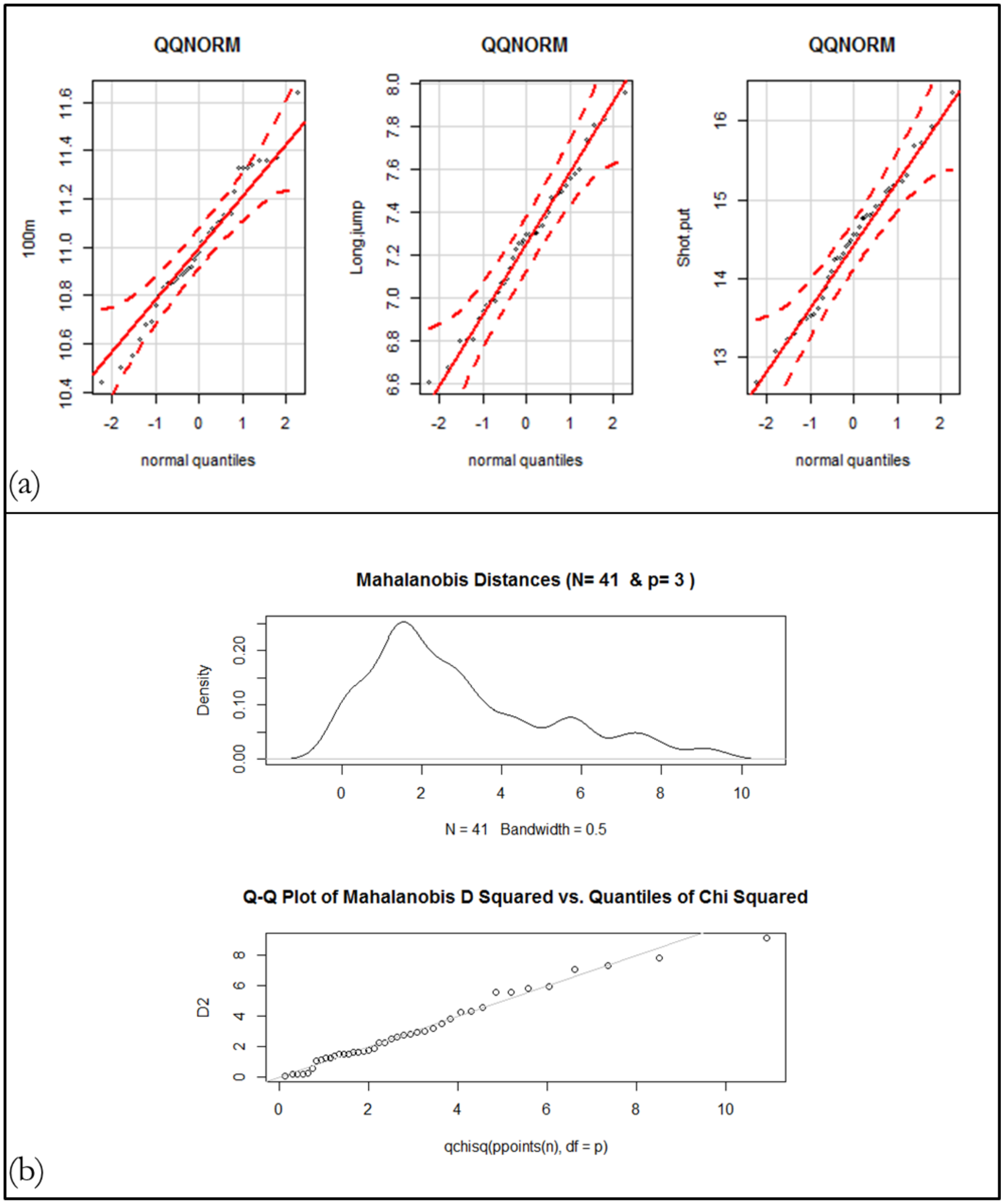
R Development Core Team. 2011. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Stacklies, W., S. H. Redesting, M. Scholz, D. Walther, and J. Selbig. 2007. pcaMethods-a bioconductor package providing PCA methods for incomplete data. Bioinformatics **23**(9): 1164-1167.

APPENDIX A. R function to examine univariate, bivariate, and multivariate normality. The function requires the package car (Fox and Weisberg 2011). User specifies the data (i.e, X) and the arguments regarding what type of examination should be performed. The default settings are to provide two plots: (1) a univariate probability plot and (2) a multi-panel plot providing the kernel density estimate and a multivariate normality plot. The user can also specify to: (1) use the Shapiro-Wilk univariate test of normality by changing the `univariate.test` argument to `TRUE` and (2) provide a multi-panel scatterplot matrix to examine bivariate normality by changing the `bivariate` argument to `TRUE`. Output from the default settings can be found in Appendix B.

```
exam.norm<-function(X,univariate=T,univariate.test=F,bivariate=F,multivariate=T){
  if(univariate==T){
    if(ncol(X)==1){par(mfrow=c(1,1))}
    if(ncol(X)>1&ncol(X)<4){par(mfrow=c(1,ncol(X)))}
    if(ncol(X)>3){par(mfrow=c(ceiling(ncol(X)/3),3))}
    for(i in 1:(ncol(X))){
      qqPlot(X[i],ylab=colnames(X)[i], xlab=paste("normal",
       "quantiles"),main="QQNORM")}}
  if(sum(univariate,bivariate,multivariate)>1){windows()}
  if(bivariate==T){
    panel.hist<- function(x, ...){
      usr <- par("usr"); on.exit(par(usr))
      par(usr = c(usr[1:2], 0, 1.5) )
      h <- hist(x,plot=F)
      breaks <- h$breaks; nB <- length(breaks)
      y <- h$counts; y <- y/max(y)
      rect(breaks[-nB], 0, breaks[-1], y, col="bisque4", ...)}
    pairs(X,diag.panel=panel.hist)}
  if(sum(univariate,bivariate,multivariate)==3){windows()}
  if(multivariate==T){
    mu<-colMeans(X)
    sig<-cov(X)
    stopifnot(mahalanobis(X,0,diag(ncol(X)))==rowSums(X*X))
    par(mfrow=c(2,1))
    D2<-mahalanobis(X,mu,sig)
    n=dim(X)[1];p=dim(X)[2]
    plot(density(D2,bw=.5),main=paste("Mahalanobis Distances (N=",as.character(n),"
      & p=",as.character(p),")"))
    qqplot(qchisq(ppoints(n),df=p),D2,main="Q-Q Plot of Mahalanobis D Squared vs.
      Quantiles of Chi Squared")
    abline(0,1,col='gray')
  }
if(univariate==F&multivariate==F&bivariate==F){print("User must specify at least
one method of examining for normality")}
  if(univariate.test==T){
    results<-vector("list",ncol(X))
    for(i in 1:(ncol(X))){
    results[[i]]<-shapiro.test(X[,i])}
    names(results)<-colnames(X)
    print(results)
  }
}
```

APPENDIX B. R graphical output from the default settings of the `exam.norm` function for the first three variables of the decathlon dataset available in the R package pcaMethods (Stacklies et al. 2007). (a) Normal probability plots and (b) a kernel density estimate plot and a multivariate normality plot.



APPENDIX C. R function to perform principal component analysis using either spectral or singular value decomposition performed by the functions `eigen` and `svd`, respectively. Output from the function consists of an object containing the eigenvalues, eigenvectors, scores, summary of output, and the standard deviations of the new components. The default settings are to use the correlation matrix (i.e., `scaled=T`) for spectral decomposition, remove all observations with missing information, and to print a summary of the results. To perform singular value decomposition the argument for method should be changed to `"svd"`. By specifying the argument `graph=T`, the function will provide two graphs: (1) a screeplot and (2) a biplot of the first two components. If there are missing values within the dataset, the user has two options: (1) `rm.na=T` (default) removes all observations with missing information or (2) `rm.na=F` replaces missing values with the mean of the variable.

```
PC<-function(X,method="eigen",scaled=T,graph=F,rm.na=T,print.results=T){
  if (any(is.na(X))){
    tmp<-X
    if(rm.na==T){X<-na.omit(data.frame(X));X<-as.matrix(X)}
    else{X[is.na(X)] = matrix(apply(X, 2, mean, na.rm = TRUE),
      ncol = ncol(X), nrow = nrow(X), byrow = TRUE)[is.na(X)]}}
  else{tmp<-X}
  if(method=="eigen"){
    if(scaled==1){X1=cor(X);X2=scale(X)}
    else{X1=cov(X);X2=scale(X,scale=F)}
    total.var<-sum(diag(cov(X2)))
    values<-eigen(X1)$values;vectors<-eigen(X1)$vectors;sdev=sqrt(values)}
  if(method=="svd"){
    if(sum(scaled,center)>1){X2<-scale(X)}else{
      if(scaled==1){X2=scale(X,center=F)}else{
        if(center==1){X2=scale(X,scale=F)}else{X2=X}}}
    total.var<-sum(diag(cov(X2)))
    var<-nrow(X2)-1
    vectors<-svd(X2)$v;sdev=svd(X2)$d/sqrt(var);values<-sdev*sdev}
  prop.var<-rep(NA,ncol(X));cum.var<-rep(NA,ncol(X));scores<-X2%*%vectors
  namex<-as.character(1:ncol(X));scorenames<-rep(NA,ncol(X))
  for(i in 1:(ncol(X))){
    scorenames[i]<-do.call(paste,c("PC",as.list(namex[i]),sep=""))}
  colnames(scores)<-scorenames
  rownames(vectors)<-colnames(X);colnames(vectors)<-scorenames
  for(i in 1:ncol(X)){prop.var[i]<-var(scores[,i])/total.var}
  for(i in 1:ncol(X)){cum.var[i]<-sum(prop.var[1:i])}
  importance<-t(matrix(c(sdev,prop.var,cum.var),ncol=3))
  importance<-as.table(importance)
  colnames(importance)<-scorenames
  rownames(importance)<-c("Standard Deviation","Proportion of Variance","Cumulative
    Proportion")
  z<-list(values=values,vectors=vectors,scores=scores,importance=importance
    ,sdev=sdev)
  if(graph==1){
    biplot(scores[,1:2],vectors[,1:2], main="Biplot of Data",xlab=do.call
     (paste,c("PC1 (",as.list(round(z$importance[2,1]*100,2)),"%)",sep=""))
     ,ylab=do.call(paste,c("PC2
     (",as.list(round(z$importance[2,2]*100,2)),"%)",sep="")), cex=0.7)
    windows()
    screeplot(z,type='l',main='Screeplot of Components')
    abline(1,0,col='red',lty=2)}

#Function Continues on Next Page
#Continued from Previous Page
if(print.results==T){
    if(method=="eigen"){print("PCA Analysis Using Spectral Decomposition")}
    if(method=="svd"){print("PCA Analyis Using Singular Value Decomposition")}
if (any(is.na(tmp))){
      if(rm.na==T){print("Warning:One or more rows of data were omitted from
        analysis")}
      if(rm.na==F){print("Warning: Mean of the variable was used for Missing
        values")}}
    print(importance)}
  z<-list(values=values,vectors=vectors,scores=scores,importance=importance
    ,sdev=sdev)
}#End Function
```

APPENDIX D. R function to examine the differences in output from the five different methods used to do principal component analysis. The function requires the packages: clusterGeneration (Qiu and Joe 2009), mvtnorm (Genz and Bretz 2009; Genz et al. 2012), FactoMineR (Husson et al. 2012), and pcaMethods (Stacklies et al. 2007). The arguments of the function specify the number of variables (`n.var`) and number of observations (`n.obs`) of each dataset, and the argument `n.reps` specifies how many simulations should be performed. The output is a data frame of mean differences of eigenvalues, eigenvectors, and scores for each simulation.

```
sim<-function(n.var=5,n.obs=1000,n.reps=1000){
  results<-data.frame(matrix(numeric(0),n.reps,3))
  colnames(results)<-c("Values","Vectors","Scores")
  for(h in 1:n.reps){
    #Function to create data
      rPCA<-function(n.var,n.obs){
       mu<-rnorm(n.var);R<-rcorrmatrix(n.var,alphad=1)
        X<-rmvnorm(n=n.obs,mean=mu,sigma=R)
       return(X)}
       data<-rPCA(n.var=n.var,n.obs=n.obs)
    ##################################
    #Create Objects from PCA analysis#
    ##################################
    ##Analysis using metrics calculated from the function eigen
      pc1<-PC(data,scale=T,rm.na=T,print.results=F)
    #Analysis using the R function prcomp
      pc2<-prcomp(data,center=TRUE,scale=TRUE)
    #Analysis using the R function princomp
      pc3<-princomp(data,cor=TRUE)
    #Analysis using the function PCA from the package FactoMinR
      pc4<-PCA(data, scale.unit=TRUE, ncp=ncol(data), graph=F)
    #Analyis using the function pca from the package pcaMethods
      pc5<-pca(data,method="svd",scale="uv",nPcs=ncol(data))
    ############################################
    #Compare Eigenvalues from different methods#
    ############################################
      eigenvalues<-data.frame(pc1$values,pc2$sdev^2,pc3$sdev^2,pc4$eig$eigenvalue,
        pc5@sDev^2)
      colnames(eigenvalues)<-c("PC","prcomp","princomp","PCA","pca")
      rownames(eigenvalues)<-colnames(data)
      valuediff<-matrix(numeric(0),nrow(eigenvalues),ncol(eigenvalues))
      for(i in 1:ncol(data)){
        for(j in 1:ncol(eigenvalues)){
          valuediff[i,j]<-eigenvalues[i,j]-rowMeans(eigenvalues)[i]}}
      results$Values[h]<-mean(valuediff) #Should be close to Zero
    #############################################
    #Compare Eigenvectors from different methods#
    #############################################
      eigenvectors<-list(PC=pc1$vectors,prcomp=pc2$rotation,princomp=pc3$loadings
       ,PCA=pc4$svd$V,pca=pc5@loadings)
      ev<- array(0, dim=c(5,ncol(data),ncol(data)))
      ev[1,,]<-eigenvectors$PC;ev[2,,]<-eigenvectors$prcomp
      ev[3,,]<-eigenvectors$princomp;ev[4,,]<-eigenvectors$PCA
      ev[5,,]<-eigenvectors$pca
      vectordiff<-array(0, dim=c(5,ncol(data),ncol(data)))
      for(i in 1:5){
        for(j in 1:ncol(data)){
          for(k in 1:ncol(data)){
            vectordiff[i,j,k]<-abs(ev[i,j,k])-mean(abs(ev[,j,k]))}}}
      results$Vectors[h]<-mean(vectordiff) #Should be close to Zero
    #######################################
    #Compare Scores from different methods#
    #######################################
      scores<-list(PC=pc1$scores,prcomp=pc2$x,princomp=pc3$scores,PCA=pc4$ind$coord
        ,pca=pc5@scores)
      sc<-array(0,dim=c(5,nrow(data),ncol(data)))
      sc[1,,]<-scores$PC;sc[2,,]<-scores$prcomp;sc[3,,]<-scores$princomp
      sc[4,,]<-scores$PCA;sc[5,,]<-scores$pca
      scorediff<-array(0,dim=c(5,nrow(data),ncol(data)))
      for(i in 1:5){
        for(j in 1:nrow(data)){
          for(k in 1:ncol(data)){
            scorediff[i,j,k]<-abs(sc[i,j,k])-mean(abs(sc[,j,k]))}}}
      results$Scores[h]<-mean(scorediff)
  }
  return(results)
}
```