**2 - Principal Components Analysis (PCA)**

Principal Component Analysis is a dimension reduction technique that has many useful applications in data analysis/statistics. Dimension reduction is achieved by forming new variables which are linear combinations of the variables in the original data set. These linear combinations are chosen to account for as much of the original variance-covariance/correlation structure in the original variables as possible. The new variables formed by these linear combinations are uncorrelated which is desirable property.

**2.1 - Terminology and Concepts**The principal component is a linear combination of the original of the *p* variables in the data set.

The coefficients are called the loadings for the principal component . Interpretation of the loadings can be an important aspect of the PCA. First we can gain important insight into how the original variables relate to one another from them. Also examining the loadings leads to our understanding what the scores (values) for the principal component are measuring.   
  
We hopefully can capture much of the information in the original variables with a much smaller number of principal components where *k << p*.

We can use the principal components in a number of ways:

* Visualize our data in a lower dimensional space
* Look for multivariate outliers. (There are variations of PCA that are specifically designed to help identify outliers in high dimensions).
* Look for clusters of similar observations
* Understand underlying structure in our data set.
* Use the principal components in subsequent analyses, e.g. regression.
* PCA can also be used as part of a process to impute (fill in) missing values in multivariate data set.

Example 2.1: Prehistoric Goblets

Measurements were made on 25 prehistoric goblets from Thailand (Professor C.F.W. Higham, University of Otago, as taken from Manly, B.F.J. 1986. *Multivariate Statistical Methods: A Primer.* Chapman and Hall, London, 159pp.) You have been asked to help organize these goblets according to their similarities. It is believed that different cultures will likely produce pottery with different to very different characteristics. The scientist has measured the mouth width (), total width (), total height (), base width (), stem width (), and stem height () on each of the 25 goblets.

> Goblets=read.table(file.choose(),header=T)

> Goblets

MouthWidth TotalWidth TotalHeight BaseWidth StemWidth StemHeight

1 13 21 23 14 7 8

2 14 14 24 19 5 9

3 19 23 24 20 6 12

4 17 18 16 16 11 8

5 19 20 16 16 10 7

6 12 20 24 17 6 9

7 12 19 22 16 6 10

8 12 22 25 15 7 7

9 11 15 17 11 6 5

10 11 13 14 11 7 4

11 12 20 25 18 5 12

12 13 21 23 15 9 8

13 12 15 19 12 5 6

14 13 22 26 17 7 10

15 14 22 26 15 7 9

16 14 19 20 17 5 10

17 15 16 15 15 9 7

18 19 21 20 16 9 10

19 12 20 26 16 7 10

20 17 20 27 18 6 14

21 13 20 27 17 6 9

22 9 9 10 7 4 3

23 8 8 7 5 2 2

24 9 9 8 4 2 2

25 12 19 27 18 5 12

Question: Do we need all six variables/dimensions to sufficiently describe the different characteristics of these goblets?

Answer: If no, then you have an opportunity to use principal components to reduce the dimension in the data while maintaining important characteristics in the data.

**2.2 - Mathematical/Statistical Concepts**

The principal components are particular linear combinations of the columns of your data matrix. Principal components depend solely on the variance/covariance matrix (or more generally on the correlation matrix) of the original *p* variables .

A principal component analysis does not necessarily rely on the multivariate normal distribution; however, if this is assumed a PCA definitely performs better.

Consider the following linear combinations of (the data matrix) that has variance/covariance matrix 



The variance/covariance term for each linear combination is given by



The principal components are defined to be the uncorrelated linear combinations that achieve maximum variances for . In particular,

|  |  |
| --- | --- |
| First Principal Component: | The linear combination that maximizes subject to the constraint of |
| Second Principal Component: | The linear combination that maximizes subject to the constraint of   AND |
| Third Principal Component: | The linear combination that maximizes subject to the constraint of   AND AND   etc.. |

Result 1:

Let be the variance/covariance matrix associated with (the data matrix). Let be the eigenvalues / eigenvectors of where it is assumed , then the principal components are given by

|  |  |  |
| --- | --- | --- |
|  | and |  |

In R

> gob.cor

MouthWidth TotalWidth TotalHeight BaseWidth StemWidth StemHeight

MouthWidth 1.0000000 0.6234051 0.3464089 0.6748429 0.6901040 0.5875703

TotalWidth 0.6234051 1.0000000 0.8392292 0.8287898 0.5807725 0.7970192

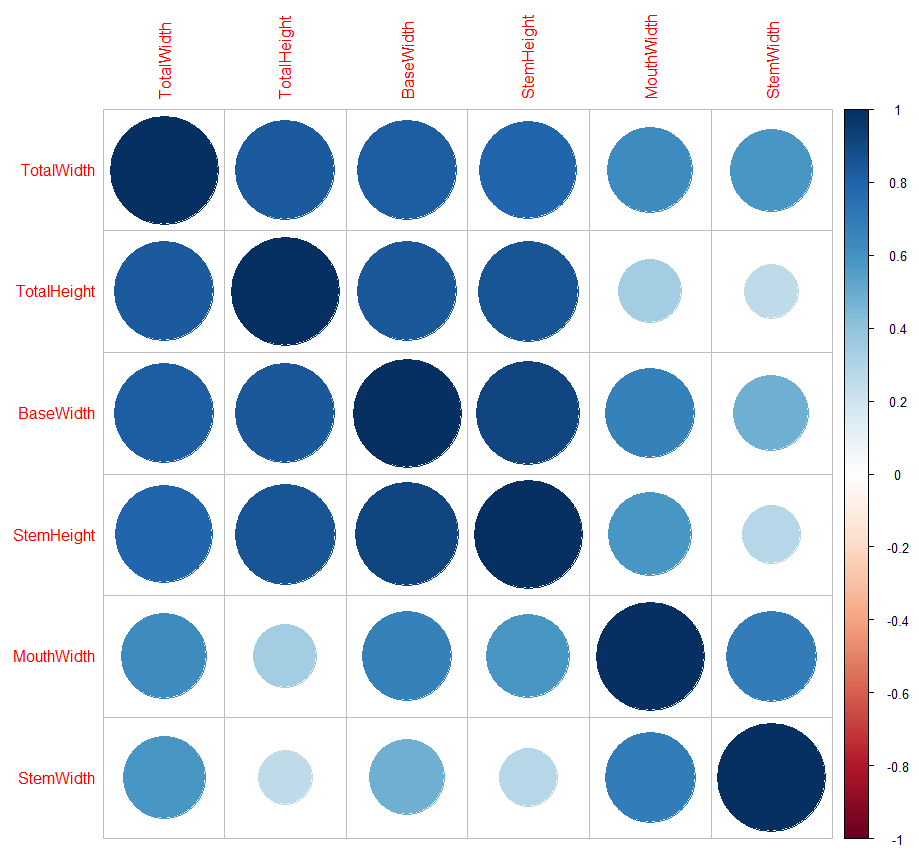
TotalHeight 0.3464089 0.8392292 1.0000000 0.8430518 0.2511584 0.8575089

BaseWidth 0.6748429 0.8287898 0.8430518 1.0000000 0.4874610 0.9101886

StemWidth 0.6901040 0.5807725 0.2511584 0.4874610 1.0000000 0.2885165

StemHeight 0.5875703 0.7970192 0.8575089 0.9101886 0.2885165 1.0000000

> corrplot(gob.corr,order=”hclust”)



Stem width and mouth width are moderately correlated and less correlated with the other four features.

Total width, total height, base width, and stem height are all moderately to strongly correlated with each other. Highlighted in yellow.

> eigen.gob = eigen(gob.cor)

> eigen.gob

$values

[1] 4.27177774 1.09217742 0.38474775 0.14239633 0.06529366 0.04360709

$vectors

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] -0.3660233 0.48592912 -0.6179335 0.32436829 -0.27835629 0.2556581

[2,] -0.4515367 -0.03412653 0.3752732 0.67427405 0.08391876 -0.4386709

[3,] -0.4111609 -0.44135161 0.3163501 -0.02019451 -0.38254463 0.6239630

[4,] -0.4618586 -0.11457532 -0.1588367 -0.54119094 -0.38182563 -0.5564635

[5,] -0.2963653 0.68277080 0.4914536 -0.35921044 0.22136144 0.1625790

[6,] -0.4381125 -0.29768029 -0.3324080 -0.13346207 0.75785442 0.1295892

> e1 = eigen.gob$vectors[,1]

> e1%\*%gob.cor%\*%e1

[,1]

[1,] 4.271778

> e2 = eigen.gob$vectors[,2]

> e2%\*%gob.cor%\*%e2

[,1]

[1,] 1.092177

Continuing with Example 2.1

Consider only the TotalWidth and TotalHeight from the goblet data.

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

Getting the variance/covariance matrix in R…

> X = cbind(TotalWidth,TotalHeight)

> var(X)

TotalWidth TotalHeight

TotalWidth 18.80667 22.11500

TotalHeight 22.11500 36.92333

Getting the eigenvalues/eigenvectors of the variance /covariance matrix in R…

> eigen(var(X))

$values

[1] 51.763256 3.966744

$vectors

[,1] [,2]

[1,] 0.5572085 -0.8303726

[2,] 0.8303726 0.5572085

|  |  |
| --- | --- |
| **Eigenvalues & Eigenvectors of** |  |

Result 2:

Let be the variance/covariance matrix associated with (the data matrix) and let be the eigenvalues / eigenvectors of where it is assumed .



Result 2 implies that  can be interpreted as the contribution to the total variance that is due to the 1st principal component (i.e. the linear combination of the original variables with maximal variance),  can be interpreted as the contribution to the total variance that is due to the 2nd principal component, etc.

Getting the linear combinations (i.e. scores) for Example 2.1.

> e1=eigen(var(X))$vectors[,1]

> e1

[1] 0.5572085 0.8303726

> e2=eigen(var(X))$vectors[,2]

> e2

[1] -0.8303726 0.5572085

> score1=X%\*%e1 🡨

> score2=X%\*%e2 🡨

|  |  |
| --- | --- |
| > cbind(score1,score2)  [,1] [,2]  [1,] 30.79995 -4.6220304  [2,] 27.72986 1.7477866  [3,] 32.74474 -5.7255671  [4,] 23.31571 -6.0313718  [5,] 24.43013 -7.6921171  [6,] 31.07311 -3.2344492  [7,] 28.85516 -3.5184936  [8,] 33.01790 -4.3379860  [9,] 22.47446 -2.9830454  [10,] 18.86893 -2.9939256  [11,] 31.90349 -2.6772408  [12,] 30.79995 -4.6220304  [13,] 24.13521 -1.8686285  [14,] 33.84828 -3.7807776  [15,] 33.84828 -3.7807776  [16,] 27.19441 -4.6329105  [17,] 21.37093 -4.9278350  [18,] 28.30883 -6.2936558  [19,] 32.73386 -2.1200323  [20,] 33.56423 -1.5628238  [21,] 33.56423 -1.5628238  [22,] 13.31860 -1.9012689  [23,] 10.27028 -2.7425217  [24,] 11.65786 -3.0156859  [25,] 33.00702 -0.7324512 |  |

These principal components are orthogonal (i.e. covariance = 0) and the total variance of these linear combinations is the same as the original data.

> round(var(cbind(score1,score2)),4)

[,1] [,2]

[1,] 51.7633 0.0000

[2,] 0.0000 3.9667

The proportion/percentage of the total variation due to the 1st principal component is . The remaining 7% is due to the 2nd principal component.

**Getting this done in R…**

Using the princomp()function to do a PC analysis in R…

> pca = princomp(X)

R returns the following…

> pca

Call:

princomp(x = X)

Standard deviations:

Comp.1 Comp.2

7.049307 1.951429

2 variables and 25 observations.

Additional output is accessible, but you need to know the names…



> names(pca) (or use attributes(pca))

[1] "sdev" "loadings" "center" "scale" "n.obs" "scores"

[7] "call"

The eigenvectors can be found under the loadings as follows

> pca$loadings

Loadings:

Comp.1 Comp.2

TotalWidth 0.557 -0.830

TotalHeight 0.830 0.557

Comp.1 Comp.2

SS loadings 1.0 1.0

Proportion Var 0.5 0.5

Cumulative Var 0.5 1.0

The summary() function computes the proportion of variance due to each principal component automatically.

> summary(pca)

Importance of components:

Comp.1 Comp.2

Standard deviation 7.0493067 1.9514288

Proportion of Variance 0.9288221 0.0711779

Cumulative Proportion 0.9288221 1.0000000

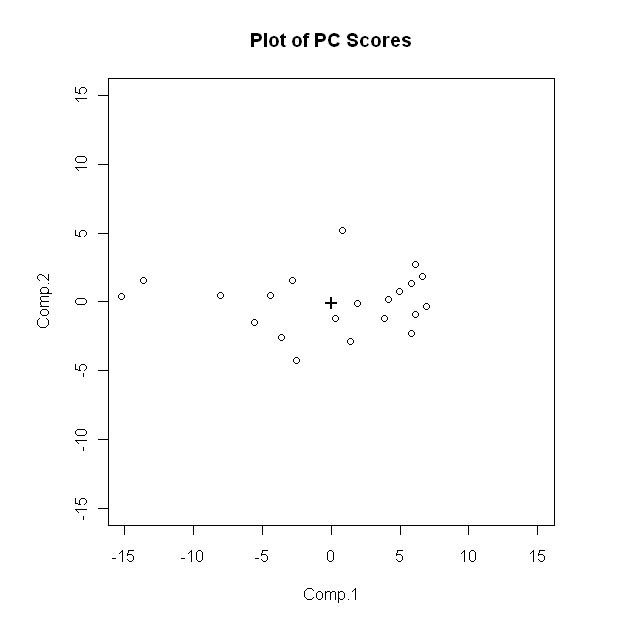
The actual linear combination values for the *centered* data can be obtained by asking for the scores…

|  |  |
| --- | --- |
| > pca$scores  Comp.1 Comp.2  [1,] 3.8865327 -1.19752382  [2,] 0.8164460 5.17229310  [3,] 5.8313223 -2.30106062  [4,] -3.5977011 -2.60686526  [5,] -2.4832842 -4.26761053  [6,] 4.1596969 0.19005729  [7,] 1.9417431 -0.09398703  [8,] 6.1044865 -0.91347950  [9,] -4.4389539 0.44146112  [10,] -8.0444888 0.43058096  [11,] 4.9900695 0.74726577  [12,] 3.8865327 -1.19752382  [13,] -2.7782087 1.55587808  [14,] 6.9348591 -0.35627103  [15,] 6.9348591 -0.35627103  [16,] 0.2809979 -1.20840399  [17,] -5.5424907 -1.50332847  [18,] 1.3954148 -2.86914926  [19,] 5.8204422 1.30447424  [20,] 6.6508148 1.86168272  [21,] 6.6508148 1.86168272  [22,] -13.5948132 1.52323759  [23,] -16.6431396 0.68198479  [24,] -15.2555585 0.40882063  [25,] 6.0936063 2.69205535 | R *centers* the data before running a PCA, which has no effect on the variance/covariance structure…    Thus, the scores are computed as follows…  1st observation scores for :  3.89 = 0.5572\*(21-17.84) + 0.8304\*(23-20.44)  = 0.5572\*3.16 + 0.8304\*2.56    -1.19 = -0.8304\*(21-17.84) + 0.5572\*(23-20.44)  = -0.8304\*3.16 + 0.5572\*2.56  2nd observation scores for :  0.82 = 0.5572\*(14-17.84) + 0.8304\*(24-20.44)  = 0.5572\*-3.84 + 0.8304\*3.56    5.17 = -0.8304\*(14-17.84) + 0.5572\*(24-20.44)  = -0.8304\*-3.84 + 0.5572\*3.56  etc… |

> plot(pca$scores,xlim=c(-15,15),ylim=c(-15,15))

> points(0,0,pch="+",cex=1.5)

> title("Plot of PC Scores")



The variance/covariance matrix of these linear combinations matches what we got above.

> round(var(pca$scores),4)

Comp.1 Comp.2

Comp.1 51.7633 0.0000

Comp.2 0.0000 3.9667

**2.3 – The Importance of Scaling/Standardizing**

Consider again all measurements recorded in the goblets dataset (Example 2.1) - the mouth width (), total width (), total height (), base width (), stem width (), and stem height ().



A quick review of image above (and the data) reveals that we’d certainly expect to have more variation than and to have more variation than , etc. These differences in variation may adversely affect our principal component analysis.

To alleviate this issue, we usually (almost always) ***standardize*** the data before performing a principal component analysis.

> meanvec=apply(Goblets,2,mean)

MouthWidth TotalWidth TotalHeight BaseWidth StemWidth StemHeight

13.28 17.84 20.44 14.60 6.36 8.12

> sigma=var(Goblets)

MouthWidth TotalWidth TotalHeight BaseWidth StemWidth StemHeight

MouthWidth 9.043333 8.13000 6.330000 8.408333 4.478333 5.548333

TotalWidth 8.130000 18.80667 22.115000 14.891667 5.435000 10.853333

TotalHeight 6.330000 22.11500 36.923333 21.225000 3.293333 16.361667

BaseWidth 8.408333 14.89167 21.225000 17.166667 4.358333 11.841667

StemWidth 4.478333 5.43500 3.293333 4.358333 4.656667 1.955000

StemHeight 5.548333 10.85333 16.361667 11.841667 1.955000 9.860000

> Std.MouthWidth=(MouthWidth-meanvec[1])/sqrt(sigma[1,1])

> Std.TotalWidth=(TotalWidth-meanvec[2])/sqrt(sigma[2,2])

> Std.TotalHeight=(TotalHeight-meanvec[3])/sqrt(sigma[3,3])

> Std.BaseWidth=(BaseWidth-meanvec[4])/sqrt(sigma[4,4])

> Std.StemWidth=(StemWidth-meanvec[5])/sqrt(sigma[5,5])

> Std.StemHeight=(StemHeight-meanvec[6])/sqrt(sigma[6,6])

Putting this together in a new matrix…

> Std.Goblets=cbind(Std.MouthWidth,Std.TotalWidth,Std.TotalHeight,   
 Std.BaseWidth,Std.StemWidth,Std.StemHeight)

The commands above are standardizing each of the original variables by converting them to z-scores by subtracting the sample mean and dividing by the sample standard deviation. This is more easily achieved by using the scale command in R.

> Std.Goblets = scale(Goblets)

As expected the mean and variance are 0 and 1, respectively. The covariance terms are

> apply(Std.Goblets,2,mean)

Std.MouthWidth Std.TotalWidth Std.TotalHeight Std.BaseWidth Std.StemWidth Std.StemHeight

0 0 0 0 0 0

> var(Std.Goblets)

Std.MouthWidth Std.TotalWidth Std.TotalHeight Std.BaseWidth Std.StemWidth Std.StemHeight

Std.MouthWidth 1.0000 0.6234 0.3464 0.6748 0.6901 0.5876

Std.TotalWidth 0.6234 1.0000 0.8392 0.8288 0.5808 0.7970

Std.TotalHeight 0.3464 0.8392 1.0000 0.8431 0.2512 0.8575

Std.BaseWidth 0.6748 0.8288 0.8431 1.0000 0.4875 0.9102

Std.StemWidth 0.6901 0.5808 0.2512 0.4875 1.0000 0.2885

Std.StemHeight 0.5876 0.7970 0.8575 0.9102 0.2885 1.0000

The covariance structure does not change in the standardization process; thus, the covariance terms in the above matrix are agree with the correlations on the original data.

> cor(Goblets)

MouthWidth TotalWidth TotalHeight BaseWidth StemWidth StemHeight

MouthWidth 1.0000 0.6234 0.3464 0.6748 0.6901 0.5876

TotalWidth 0.6234 1.0000 0.8392 0.8288 0.5808 0.7970

TotalHeight 0.3464 0.8392 1.0000 0.8431 0.2512 0.8575

BaseWidth 0.6748 0.8288 0.8431 1.0000 0.4875 0.9102

StemWidth 0.6901 0.5808 0.2512 0.4875 1.0000 0.2885

StemHeight 0.5876 0.7970 0.8575 0.9102 0.2885 1.0000

Paraphrasing the results above :

Doing a principal component analysis using the correlation matrix instead of the variance/covariance matrix is equivalent to first standardizing your data (to alleviate issues with differences in scale amongst the ) and running a PC analysis on the standardized variables. In JMP for example you can choose between performing PCA on the variance/covariance matrix (unstandardized) or PCA on the correlation matrix (standardized).

Example 2.1: Goblets (cont’d)

All measurements recorded -- the mouth width (), total width (), total height (), base width (), stem width (), and stem height () on each of the 25 goblets.



Doing the PC analysis in R, using the cor=TRUEoption.

> pca=princomp(Goblets,cor=TRUE)

Using the summary() function to determine the proportion of variance due to each PC

> summary(pca)

Importance of components:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6

Standard deviation 2.066828 1.0450729 0.62028038 0.37735439 0.25552624 0.208823116



Proportion of Variance 0.711963 0.1820296 0.06412462 0.02373272 0.01088228 0.007267849

Cumulative Proportion 0.711963 0.8939925 0.95811715 0.98184987 0.99273215 1.000000000

A **scree plot** is a plot that is used to help determine how many principal components should be retained. In the above, we see that last three principal components add very little to our understanding of the total variation in the Goblet data.

Getting a Scree Plot in R...

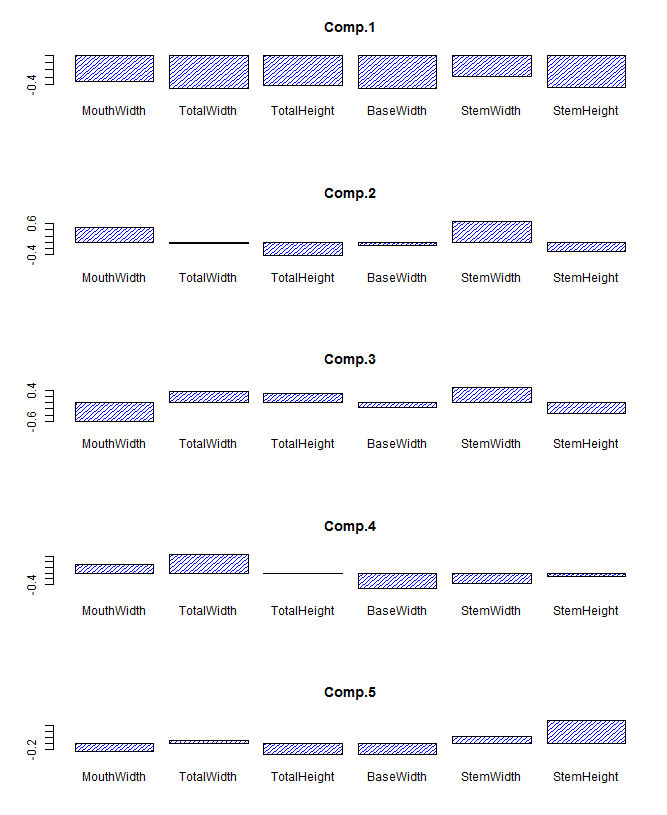
> plot(pca,type="bar",main="Scree Plot for Goblet Data")

A line version of this plot can be obtained in R by changing the type to line…

> plot(pca,type="line",main="Scree Plot for Goblet Data")



How are the principal component scores computed, i.e. what are the defining linear combinations of the original ?



> pca$loadings

Loadings:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6

MouthWidth -0.366 0.486 0.618 0.324 -0.278 0.256

TotalWidth -0.452 -0.375 0.674 -0.439

TotalHeight -0.411 -0.441 -0.316 -0.383 0.624

BaseWidth -0.462 -0.115 0.159 -0.541 -0.382 -0.556

StemWidth -0.296 0.683 -0.491 -0.359 0.221 0.163

StemHeight -0.438 -0.298 0.332 -0.133 0.758 0.130

> loadplot(pca$loadings)

**Questions:**

1. What is the make-up of the 1st principal component?
2. What is the make-up of the 2nd principal component?

Plotting the scores in R

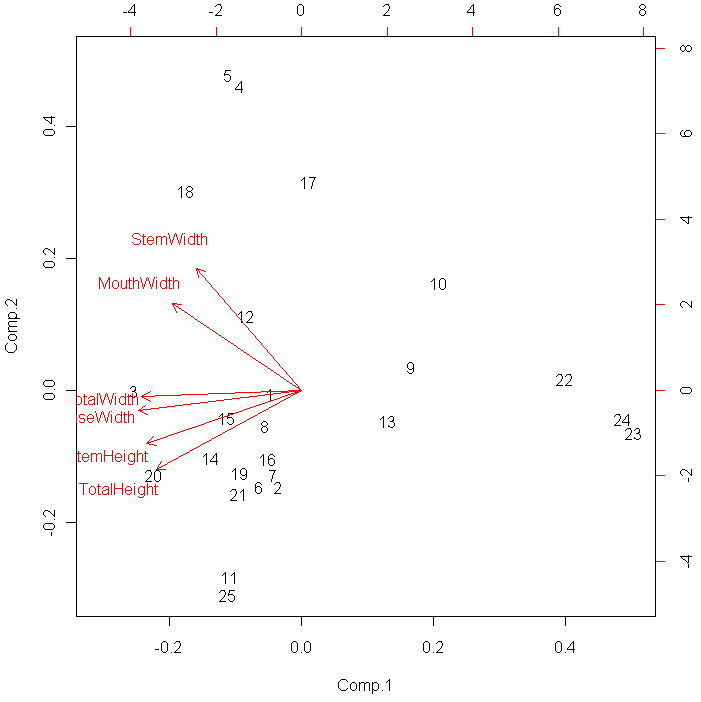
> plot(pca$scores,main="Plot of PC1 vs PC2 for Goblet Data")

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

The following plot is called a **biplot**. This plot shows the first couple PC’s and includes the direction to which observations are being pulled due to each of the variables.

In R…

> biplot(pca)



**2.4 – Principal Components Using Singular Value Decomposition (SVD)**

As we have seen above the principal components can be derived by performing a spectral decomposition (i.e. eigenanalysis of the variance/covariance matrix or the correlation matrix). As an alternative we consider Singular Value Decomposition (SVD).

Singular value decomposition will decompose a data matrix into a product of three matrices.

where,

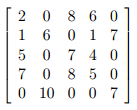
.

matrix containing the singular values in a submatrix.   
 The singular values are the square roots of the non-zero eigenvalues of .

.

Typically the first few singular values are large relative to the rest. We can achieve an approximation to the original matrix by retaining the largest singular values and their corresponding eigenvectors contained in (. Below is an example of a SVD decomposition of a matrix and a lower dimensional approximation to the matrix using the method described above.

Consider the matrix below:



we will decompose this matrix using SVD in R using the command svd.

> X = matrix(scan(),nrow=5,ncol=5,byrow=T)

1: 2 0 8 6 0

6: 1 6 0 1 7

11: 5 0 7 4 0

16: 7 0 8 5 0

21: 0 10 0 0 7

26:

Read 25 items

> Xsvd = svd(X)

> Xsvd

$d

[1] 17.9183709 15.1713719 3.5640020 1.9842282 0.3495557

$u

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

[1,] -0.54225536 0.06499573 0.82161708 0.10574661 -0.12448979

[2,] -0.10181247 -0.59346055 -0.11255162 0.78812338 0.06026999

[3,] -0.52495325 0.05938171 -0.21296861 -0.11574223 0.81372354

[4,] -0.64487038 0.07040626 -0.50874368 -0.05990271 -0.56282918

[5,] -0.06449519 -0.79692967 0.09000966 -0.59219473 -0.04412631

$v

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

[1,] -0.46461713 0.02150651 -0.86850856 0.0007995538 -0.1713494

[2,] -0.07008599 -0.75998796 0.06307148 -0.6013456703 -0.2278412

[3,] -0.73509354 0.09879712 0.28400852 -0.2234845728 0.5650402

[4,] -0.48439167 0.02544740 0.39886566 0.3326838129 -0.7035231

[5,] -0.06496983 -0.64151954 -0.04427431 0.6912010359 0.3232839

> U = Xsvd$u

> D = diag(Xsvd$d)

> D

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

[1,] 17.91837 0.00000 0.000000 0.000000 0.0000000

[2,] 0.00000 15.17137 0.000000 0.000000 0.0000000

[3,] 0.00000 0.00000 3.564002 0.000000 0.0000000

[4,] 0.00000 0.00000 0.000000 1.984228 0.0000000

[5,] 0.00000 0.00000 0.000000 0.000000 0.3495557

> V = Xsvd$v

> U%\*%D%\*%t(V)

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

[1,] 2.000000e+00 -1.913400e-15 8.000000e+00 6.000000e+00 -8.552187e-16

[2,] 1.000000e+00 6.000000e+00 6.401130e-16 1.000000e+00 7.000000e+00

[3,] 5.000000e+00 1.762479e-15 7.000000e+00 4.000000e+00 7.216450e-16

[4,] 7.000000e+00 -1.110223e-16 8.000000e+00 5.000000e+00 -8.326673e-17

[5,] 5.306519e-15 1.000000e+01 -1.916869e-15 -5.349887e-15 7.000000e+00

Notice that the full singular value decomposition !

We can alternatively complete the SVD of by finding completing the eigenanalysis of & and then using the results to form This process is shown below.

> XXt = X%\*%t(X)  
> XXt

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

[1,] 104 8 90 108 0

[2,] 8 87 9 12 109

[3,] 90 9 90 111 0

[4,] 108 12 111 138 0

[5,] 0 109 0 0 149

> XtX = t(X)%\*%X  
> XtX

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

[1,] 79 6 107 68 7

[2,] 6 136 0 6 112

[3,] 107 0 177 116 0

[4,] 68 6 116 78 7

[5,] 7 112 0 7 98

> XXteigen = eigen(XXt)  
> XXteigen

$values

[1] 321.0680142 230.1705247 12.7021105 3.9371614 0.1221892

$vectors

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

[1,] -0.54225536 -0.06499573 0.82161708 -0.10574661 -0.12448979

[2,] -0.10181247 0.59346055 -0.11255162 -0.78812338 0.06026999

[3,] -0.52495325 -0.05938171 -0.21296861 0.11574223 0.81372354

[4,] -0.64487038 -0.07040626 -0.50874368 0.05990271 -0.56282918

[5,] -0.06449519 0.79692967 0.09000966 0.59219473 -0.04412631

> U = XXteigen$vectors

> D = diag(sqrt(XXteigen$values))

> U

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

[1,] -0.54225536 -0.06499573 0.82161708 -0.10574661 -0.12448979

[2,] -0.10181247 0.59346055 -0.11255162 -0.78812338 0.06026999

[3,] -0.52495325 -0.05938171 -0.21296861 0.11574223 0.81372354

[4,] -0.64487038 -0.07040626 -0.50874368 0.05990271 -0.56282918

[5,] -0.06449519 0.79692967 0.09000966 0.59219473 -0.04412631

Notice that the matrix obtained is the “same” with the exception that some of the columns of are negated from those obtained from the SVD. This is presents a problem when checking that . This happens because if is an eigenvector of a matrix then so is .  
> D

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

[1,] 17.91837 0.00000 0.000000 0.000000 0.0000000

[2,] 0.00000 15.17137 0.000000 0.000000 0.0000000

[3,] 0.00000 0.00000 3.564002 0.000000 0.0000000

[4,] 0.00000 0.00000 0.000000 1.984228 0.0000000

[5,] 0.00000 0.00000 0.000000 0.000000 0.3495557

> XtXeigen = eigen(XtX)

> XtXeigen

$values

[1] 321.0680142 230.1705247 12.7021105 3.9371614 0.1221892

$vectors

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

[1,] -0.46461713 -0.02150651 0.86850856 0.0007995538 0.1713494

[2,] -0.07008599 0.75998796 -0.06307148 -0.6013456703 0.2278412

[3,] -0.73509354 -0.09879712 -0.28400852 -0.2234845728 -0.5650402

[4,] -0.48439167 -0.02544740 -0.39886566 0.3326838129 0.7035231

[5,] -0.06496983 0.64151954 0.04427431 0.6912010359 -0.3232839

> V = XtXeigen$vectors

> V

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

[1,] -0.46461713 -0.02150651 0.86850856 0.0007995538 0.1713494

[2,] -0.07008599 0.75998796 -0.06307148 -0.6013456703 0.2278412

[3,] -0.73509354 -0.09879712 -0.28400852 -0.2234845728 -0.5650402

[4,] -0.48439167 -0.02544740 -0.39886566 0.3326838129 0.7035231

[5,] -0.06496983 0.64151954 0.04427431 0.6912010359 -0.3232839

Again we notice that the matrix obtained is the “same” with the exception that some of the columns of are negated from those obtained from the SVD. As stated above this happens because if is an eigenvector of a matrix then so is .

Thus when conducting a SVD in R, we use the svd function in R rather than doing the eigenanalyses of and directly, which actually is far easier anyway.

**Approximating the Original Data Matrix (i.e. reducing dimension using PCA)**

As the first two or three singular values are much larger than the others we can use these singular values along with their associated eigenvectors in to approximate . Below I show the approximation to based upon 2 and 3 singular values and the associated eigenvectors.

> Xsvd

$d

[1] 17.9183709 15.1713719 3.5640020 1.9842282 0.3495557

$u

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

[1,] -0.54225536 0.06499573 0.82161708 0.10574661 -0.12448979

[2,] -0.10181247 -0.59346055 -0.11255162 0.78812338 0.06026999

[3,] -0.52495325 0.05938171 -0.21296861 -0.11574223 0.81372354

[4,] -0.64487038 0.07040626 -0.50874368 -0.05990271 -0.56282918

[5,] -0.06449519 -0.79692967 0.09000966 -0.59219473 -0.04412631

$v

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

[1,] -0.46461713 0.02150651 -0.86850856 0.0007995538 -0.1713494

[2,] -0.07008599 -0.75998796 0.06307148 -0.6013456703 -0.2278412

[3,] -0.73509354 0.09879712 0.28400852 -0.2234845728 0.5650402

[4,] -0.48439167 0.02544740 0.39886566 0.3326838129 -0.7035231

[5,] -0.06496983 -0.64151954 -0.04427431 0.6912010359 0.3232839

Two dimensional approximation to

> U = Xsvd$u

> V = Xsvd$v

> D = diag(Xsvd$d)

> U2 = U[,1:2]

> D2 = diag(Xsvd$d[1:2])

> V2 = V[,1:2]

> Xapprox2 = U2%\*%D2%\*%t(V2)

> Xapprox2

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

[1,] 4.5355816 -0.068425900 7.2398346 4.7316035 -0.001317455

[2,] 0.6539711 6.970494560 0.4515103 0.6545638 5.894517555

[3,] 4.3897067 -0.025424351 7.0035221 4.5792624 0.033179993

[4,] 5.3916357 -0.001942943 8.5995564 5.6243404 0.065482963

[5,] 0.2769094 9.269641750 -0.3449983 0.2521144 7.831384884

> X

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

[1,] 2 0 8 6 0

[2,] 1 6 0 1 7

[3,] 5 0 7 4 0

[4,] 7 0 8 5 0

[5,] 0 10 0 0 7

> X – Xapprox2

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

[1,] -2.5355816 0.068425900 0.760165410 1.2683965 0.001317455

[2,] 0.3460289 -0.970494560 -0.451510307 0.3454362 1.105482445

[3,] 0.6102933 0.025424351 -0.003522082 -0.5792624 -0.033179993

[4,] 1.6083643 0.001942943 -0.599556442 -0.6243404 -0.065482963

[5,] -0.2769094 0.730358250 0.344998304 -0.2521144 -0.831384884

Three dimensional approximation to

> U3 = U[,1:3]

> D3 = diag(Xsvd$d[1:3])

> V3 = V[,1:3]

> Xapprox3 = U3%\*%D3%\*%t(V3)

> Xapprox3

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

[1,] 1.992375772 0.1162628 8.0714811 5.8995799 -0.1309635

[2,] 1.002359586 6.9451944 0.3375848 0.4945652 5.9122775

[3,] 5.048922544 -0.0732969 6.7879538 4.2765151 0.0667851

[4,] 6.966383726 -0.1163018 8.0846026 4.9011318 0.1457595

[5,] -0.001703482 9.2898747 -0.2538899 0.3800684 7.8171819

> X

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

[1,] 2 0 8 6 0

[2,] 1 6 0 1 7

[3,] 5 0 7 4 0

[4,] 7 0 8 5 0

[5,] 0 10 0 0 7

> X - Xapprox3

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

[1,] 0.007624228 -0.1162628 -0.07148109 0.10042011 0.1309635

[2,] -0.002359586 -0.9451944 -0.33758478 0.50543484 1.0877225

[3,] -0.048922544 0.0732969 0.21204622 -0.27651513 -0.0667851

[4,] 0.033616274 0.1163018 -0.08460256 0.09886822 -0.1457595

[5,] 0.001703482 0.7101253 0.25388990 -0.38006837 -0.8171819

We can see the three dimensional approximation to is definitely better than the two dimensional approximation. SVD can be used to reduce the dimensionality of data set and examining the components (rows and columns) of the and matrices can help gain insight to underlying structure in the data matrix . We will not dig into the interpretation details at this point. SVD can also be used in the process of filling in or imputing the missing values in a data matrix.

Example 2.1: Prehistoric Goblets (cont’d)

In the SVD discussion above we saw an example where we considered two lower dimensional approximations of a matrix by using the first few singular values along with the associated rows and columns of the and matrices. Below we consider a SVD of the scaled data matrix for the goblets.

> Xs = scale(Goblets)

> Xs.svd = svd(Xs)

> Xs.svd

$d

[1] 10.125348 5.119791 3.038741 1.848651 1.251818 1.023020

$u

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 0.046658409 -0.004997796 0.21249724 -0.232359182 -0.01428663 0.0422986277

[2,] 0.035077873 -0.145985775 -0.28514098 0.496016995 0.55733342 0.1545544675

[3,] 0.253642407 -0.001163343 -0.40913245 -0.322994897 0.20119584 -0.2564515420

[4,] 0.093386799 0.461561581 0.01120617 0.275475873 -0.20472352 0.0007299643

[5,] 0.110651435 0.478327330 -0.10719162 -0.122465381 0.18699016 -0.1448075956

[6,] 0.064280588 -0.145728752 0.12114323 0.056808062 0.08751792 -0.2687046765

[7,] 0.043402442 -0.128933524 0.03617963 0.089653070 -0.26402295 -0.1989520107

[8,] 0.055515724 -0.053354437 0.36508218 -0.206857113 0.26331148 -0.1105559762

[9,] -0.167108680 0.036175773 0.14149224 0.007194744 0.06832315 0.0662955452

[10,] -0.207938968 0.162126057 0.14292326 0.237065921 0.05922116 -0.0037689731

[11,] 0.109747592 -0.282665834 -0.05379710 0.108192078 -0.28502732 -0.2522353463

[12,] 0.084795123 0.113199986 0.34977444 0.018386032 -0.10455945 0.0583051319

[13,] -0.130497391 -0.046353575 -0.01426302 -0.043953262 0.20561097 0.1855596055

[14,] 0.137576521 -0.102331850 0.18485123 -0.053119285 -0.04361867 -0.0686231047

[15,] 0.113799316 -0.041451375 0.17729834 -0.275770833 0.07588876 0.2367044815

[16,] 0.051524175 -0.104638051 -0.22089058 -0.050024335 -0.06115685 -0.4384257062

[17,] -0.009820235 0.316018288 -0.03007857 0.284847152 -0.08891099 -0.1244404624

[18,] 0.175440480 0.302694317 -0.18964104 -0.220450409 -0.12375778 0.2051912981

[19,] 0.093980135 -0.125417921 0.20813386 0.102782658 -0.16026229 0.1773138428

[20,] 0.224340122 -0.128465729 -0.35236584 -0.043920802 -0.28227589 0.5183505135

[21,] 0.096349494 -0.156727504 0.10491976 0.003854217 0.31233377 0.1155213216

[22,] -0.398351803 0.016621285 -0.04381299 0.107241383 -0.08383008 0.0878971203

[23,] -0.503629424 -0.065122918 -0.14589129 -0.100092180 -0.08373264 -0.1225152377

[24,] -0.485651937 -0.043883993 -0.15528698 -0.311403790 -0.04857464 0.0933668908

[25,] 0.112829803 -0.309502241 -0.04800909 0.195893281 -0.16898695 0.0473918209

$v

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 0.3660233 0.48592912 -0.6179335 -0.32436829 0.27835629 0.2556581

[2,] 0.4515367 -0.03412653 0.3752732 -0.67427405 -0.08391876 -0.4386709

[3,] 0.4111609 -0.44135161 0.3163501 0.02019451 0.38254463 0.6239630

[4,] 0.4618586 -0.11457532 -0.1588367 0.54119094 0.38182563 -0.5564635

[5,] 0.2963653 0.68277080 0.4914536 0.35921044 -0.22136144 0.1625790

[6,] 0.4381125 -0.29768029 -0.3324080 0.13346207 -0.75785442 0.1295892

> Goblets.pc = princomp(Xs)

> Goblets.pc$loadings

Do you see any similarities between the principal component analysis on the correlation matrix and the SVD of the scaled data matrix ***X***?

Loadings:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6

MouthWidth -0.366 0.486 -0.618 0.324 -0.278 0.256

TotalWidth -0.452 0.375 0.674 -0.439

TotalHeight -0.411 -0.441 0.316 -0.383 0.624

BaseWidth -0.462 -0.115 -0.159 -0.541 -0.382 -0.556

StemWidth -0.296 0.683 0.491 -0.359 0.221 0.163

StemHeight -0.438 -0.298 -0.332 -0.133 0.758 0.130

We see that the columns of the V matrix are the same as the PCA loadings, modulo the sign. Thus they are essentially equivalent. Using the SVD approach for conducting a PCA has advantages in certain contexts. The example below will illustrate a situation where the SVD approach is much preferred.

Example 2.2 – Image Compression and Processing

Consider the following pixel image.

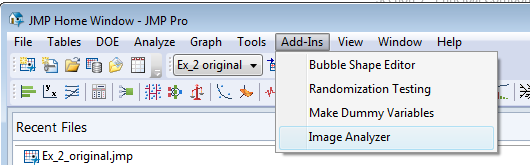
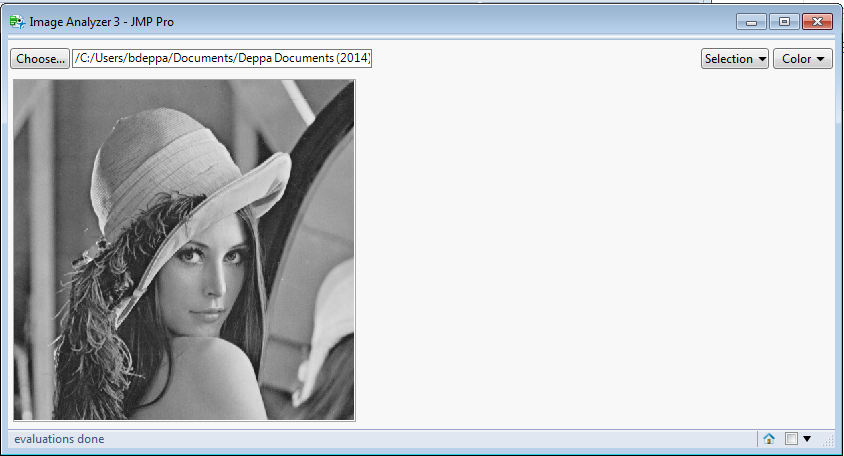


The information contained in this image can be stored in a data matrix where the elements of the matric contain the gray scale intensity of the pixel. The intensity is value between 0 (black) and 1 (white), thus intensities closer 1 indicate lighter pixels and those closer to 0 are dark pixels. We can perform a singular value decomposition of the scaled intensity matrix to produce a lower dimensional (less detailed) version of the image. This is called *digital image compression*.

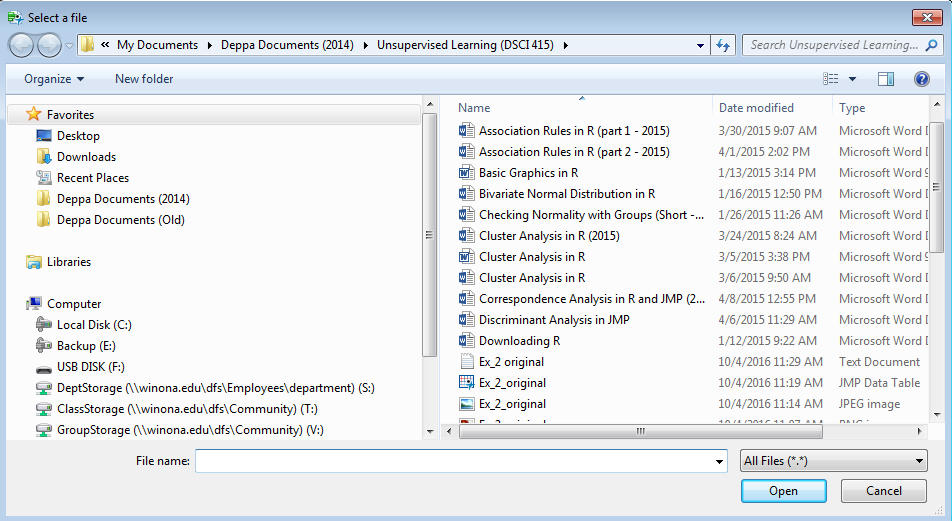
Although there are R packages that allow us to read in image files directly we will use the *JMP Image Analyzer* add-in script to read in the image file. You can down it here:

<https://community.jmp.com/docs/DOC-7181> Use **File > Open** to navigate to the add-in file once you have downloaded it.

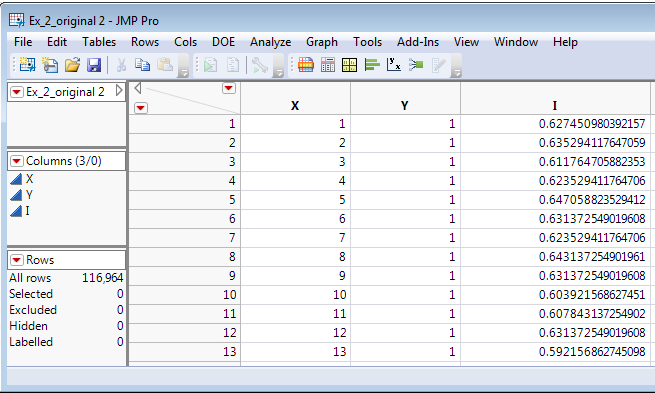
Once you have an image file in .JPEG format you can then select Image Analyzer from the Add-In drop-down menu to open any .JPEG or .jpg image file.

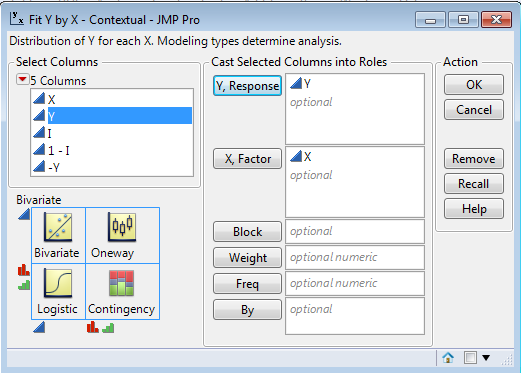
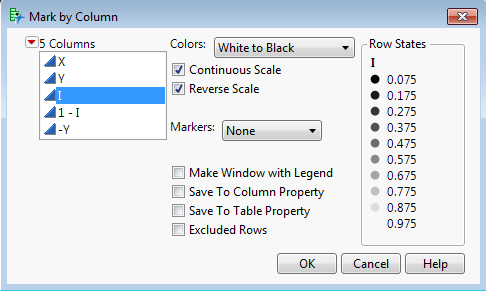
Choose the .JPEG image file.



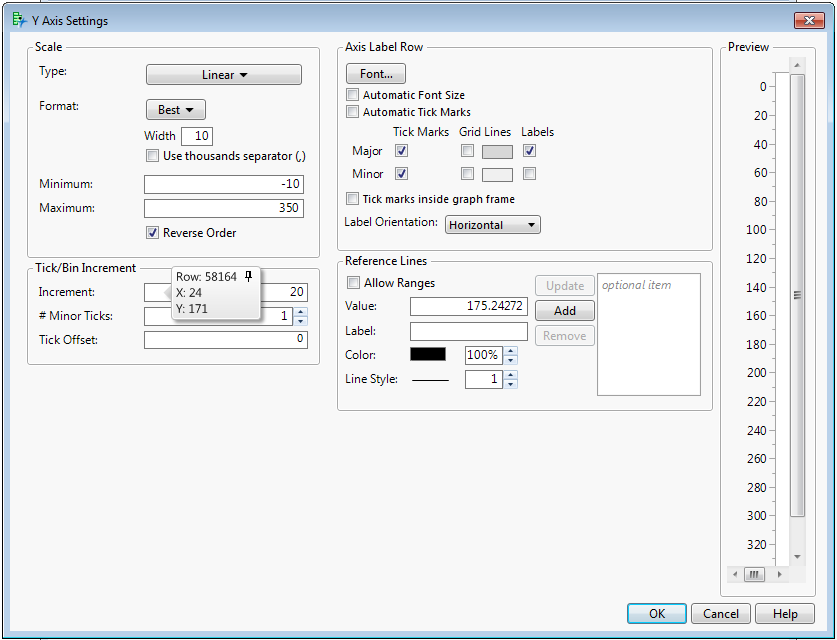
A portion of the resulting file is shown below:

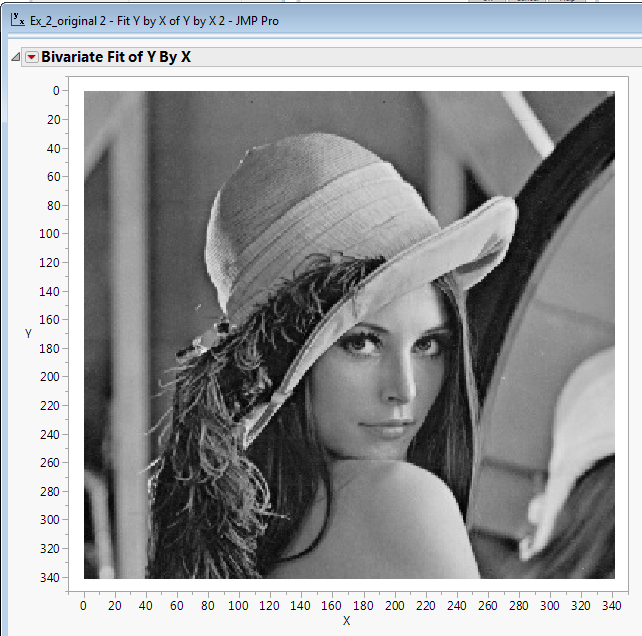


We can use **Analyze > Fit Y by X** to view the image by color coding the pixels using **Rows > Color or Marker by Column**.

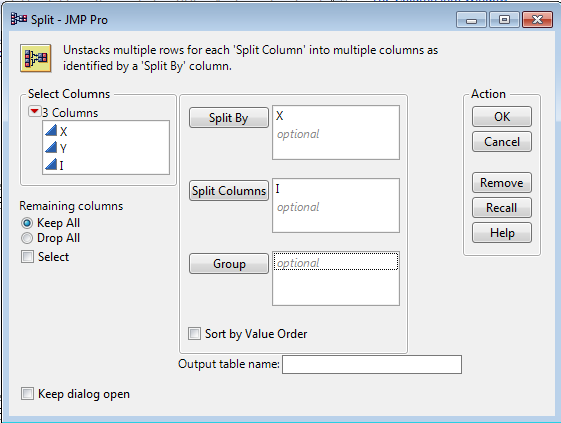
We can reverse the Y-axis by right-clicking on it and selecting **Reverse Order**.

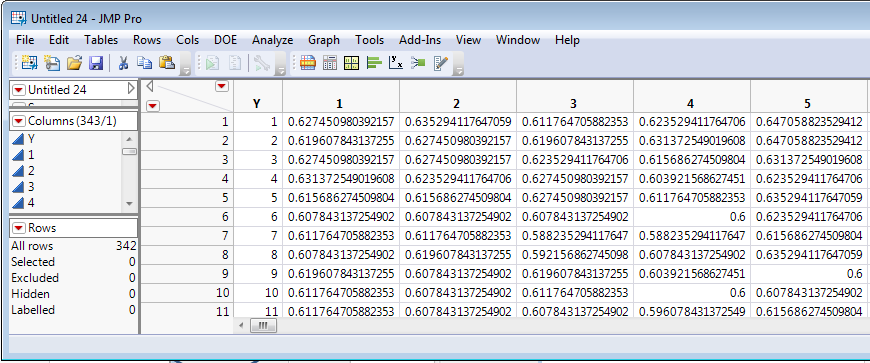


This was a lot of effort, but it is worthwhile in the sense we now understand how a digital image is really just a data matrix with numeric variables representing the details for each pixel. If we had a color image we can measure the Red, Green, and Blue (RGB) intensities of the cells along with Intensity, Hue, Lightness and Saturation. Thus for each pixel in the image we obtain seven numeric values. Thus image information would need to be stored in an array (i.e. multiple matrices) consisting of seven separate matrices, one for each pixel attribute.

We can then use the **Tables > Split** command to turn the image into a data matrix.



The resulting image matrix can then be read into R after saving it as .csv or .txt file using comma delimiters.



> Ximage = read.table(file.choose(),header=T,sep=”,”)

> names(X)

  
> X = as.matrix(Ximage[,-1])



> x = seq(1:342)

> y = seq(1:342)

> image(x,y,X,col=gray((0:341)/341))

SIDEWAYS!! Ugh….

This can be fixed using the commands below but the details are not that important.

> X = apply(X,2,rev)

> image(x,y,t(X),col=gray((0:341)/341))



Finally we will consider using SVD to compress the detail of the image using the same procedure we did with the toy SVD example above.

> Xsvd = svd(X)

> Xsvd$d

[1] 1.699e+02 2.771e+01 2.149e+01 1.692e+01 1.540e+01 1.451e+01

[7] 1.201e+01 1.074e+01 8.851e+00 8.296e+00 6.990e+00 6.396e+00

[13] 6.245e+00 6.099e+00 5.740e+00 5.602e+00 5.485e+00 4.854e+00

[19] 4.580e+00 4.448e+00 4.328e+00 4.133e+00 3.804e+00 3.526e+00

[25] 3.486e+00 3.411e+00 3.214e+00 3.118e+00 3.040e+00 2.989e+00

[31] 2.793e+00 2.757e+00 2.717e+00 2.707e+00 2.636e+00 2.589e+00

First 100 singular values of the SVD decomposition of the image matrix.

[37] 2.534e+00 2.505e+00 2.429e+00 2.402e+00 2.334e+00 2.232e+00

[43] 2.221e+00 2.175e+00 2.120e+00 2.075e+00 2.004e+00 1.990e+00

[49] 1.950e+00 1.921e+00 1.889e+00 1.842e+00 1.798e+00 1.777e+00

[55] 1.737e+00 1.690e+00 1.676e+00 1.665e+00 1.642e+00 1.610e+00

[61] 1.580e+00 1.567e+00 1.541e+00 1.511e+00 1.485e+00 1.435e+00

[67] 1.408e+00 1.374e+00 1.367e+00 1.359e+00 1.316e+00 1.275e+00

[73] 1.256e+00 1.240e+00 1.227e+00 1.202e+00 1.174e+00 1.167e+00

[79] 1.141e+00 1.132e+00 1.120e+00 1.103e+00 1.093e+00 1.075e+00

[85] 1.058e+00 1.033e+00 1.024e+00 1.013e+00 1.002e+00 9.871e-01

[91] 9.848e-01 9.606e-01 9.526e-01 9.402e-01 9.111e-01 9.053e-01

[97] 8.872e-01 8.810e-01 8.646e-01 8.508e-01

> dim(Xsvd$u)

[1] 342 342

> dim(Xsvd$v)

[1] 342 342

By extracting the singular values () and their corresponding vectors in and we can construct a lower dimensional representation of the image, i.e. a compressed image.

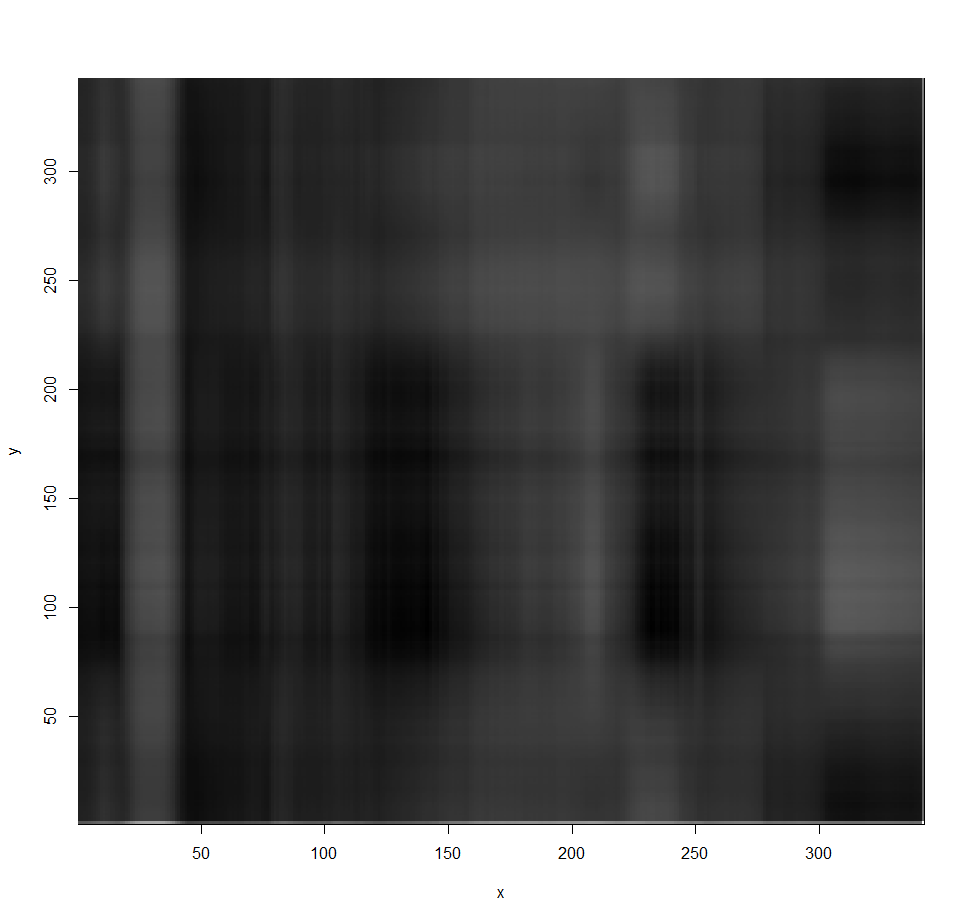
k = 2

> D2 = diag(Xsvd$d[1:2])

> U2 = Xsvd$u[,1:2]

> V2 = Xsvd$v[,1:2]

> Xapprox2 = U2%\*%D2%\*%t(V2)

> image(x,y,t(Xapprox2),col=gray((0:341)/341))  


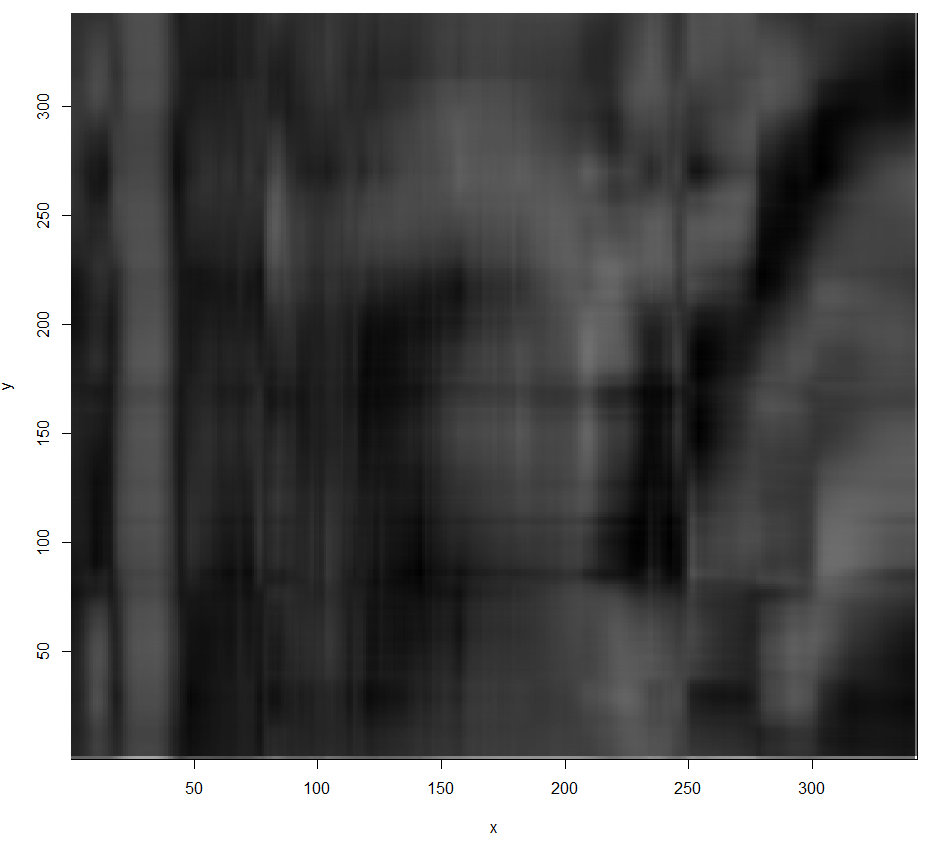
k = 5

> D5 = diag(Xsvd$d[1:5])

> U5 = Xsvd$u[,1:5]

> V5 = Xsvd$v[,1:5]

> Xapprox5 = U5%\*%D5%\*%t(V5)

> image(x,y,t(Xapprox5),col=gray((0:341)/341))  


This is rather tedious as the commands for the and dimensional representations were essentially identical except for the number of dimensions to extract. We can easily write function that does all of this for a given .

image.svd = function(Xsvd,k=10){

Dk = diag(Xsvd$d[1:k])

Uk = Xsvd$u[,1:k]

Vk = Xsvd$v[,1:k]

Xk = Uk%\*%Dk%\*%t(Vk)

p = dim(Xsvd$u)[1]

x = seq(1:p)

y = x

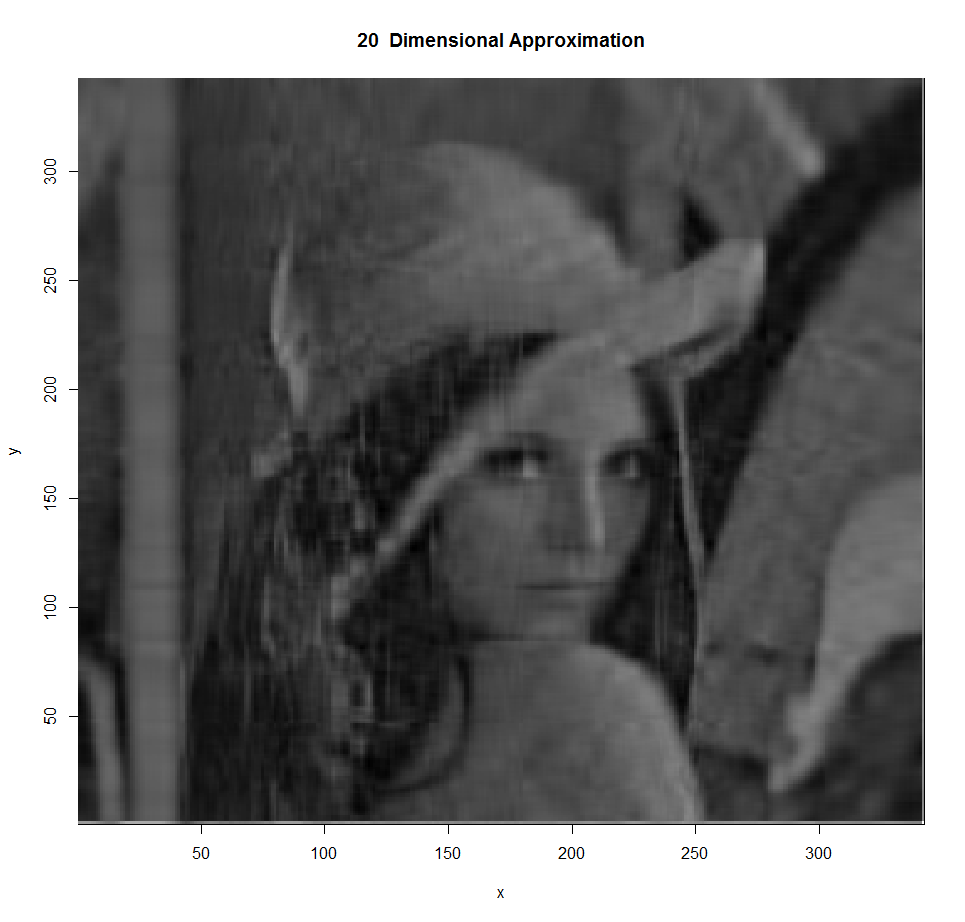
image(x,y,t(Xk),col=gray((0:(p-1))/(p-1)))

title(paste(k," Dimensional Approximation"))

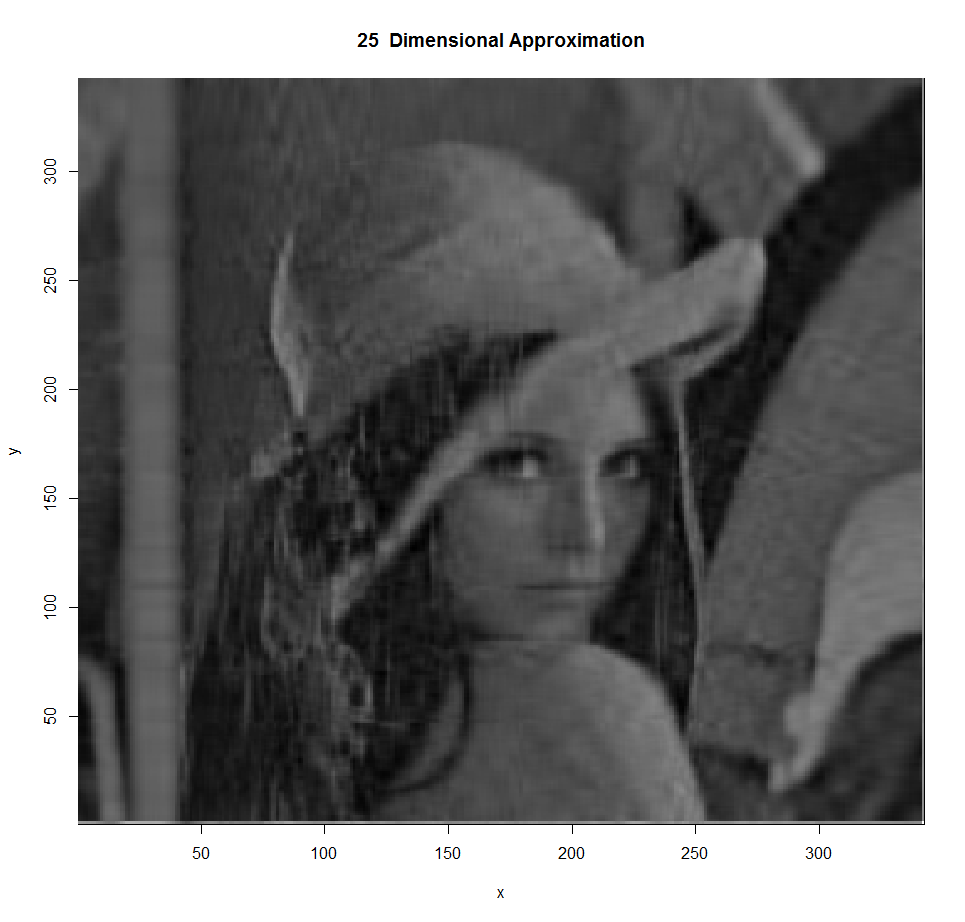
}

k = 20

> image.svd(Xsvd,k=20)

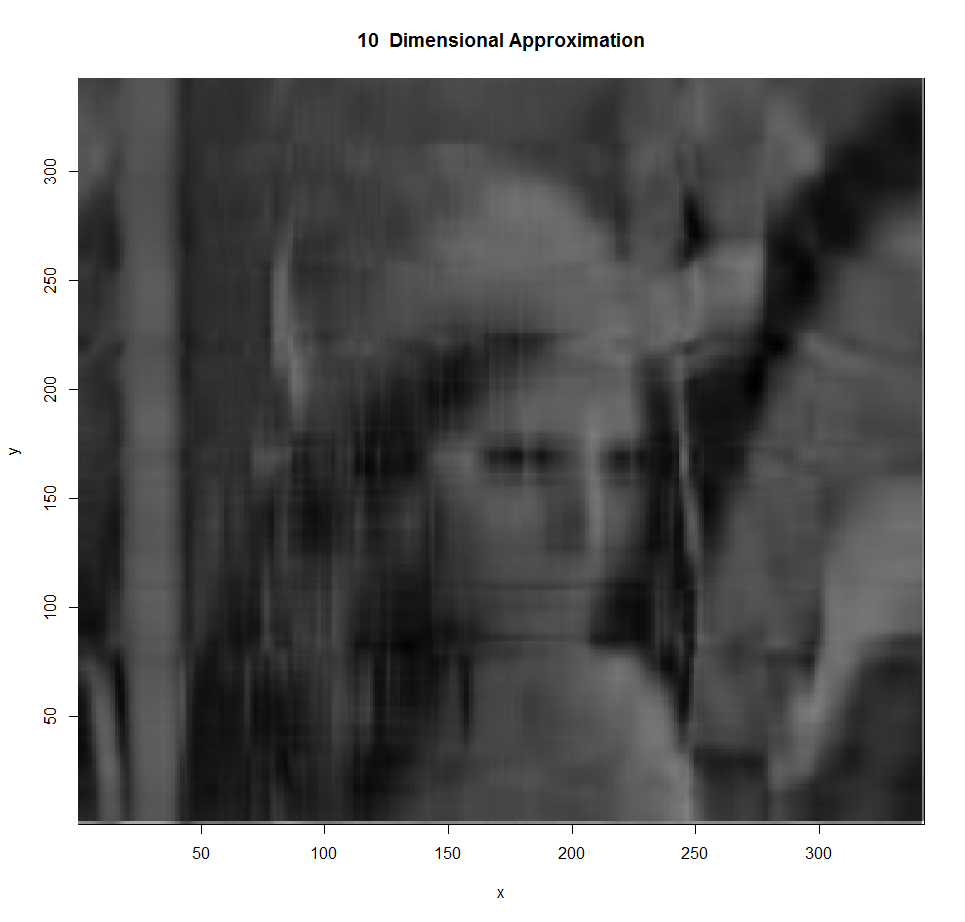


k = 25  
> image.svd(Xsvd,k=25)



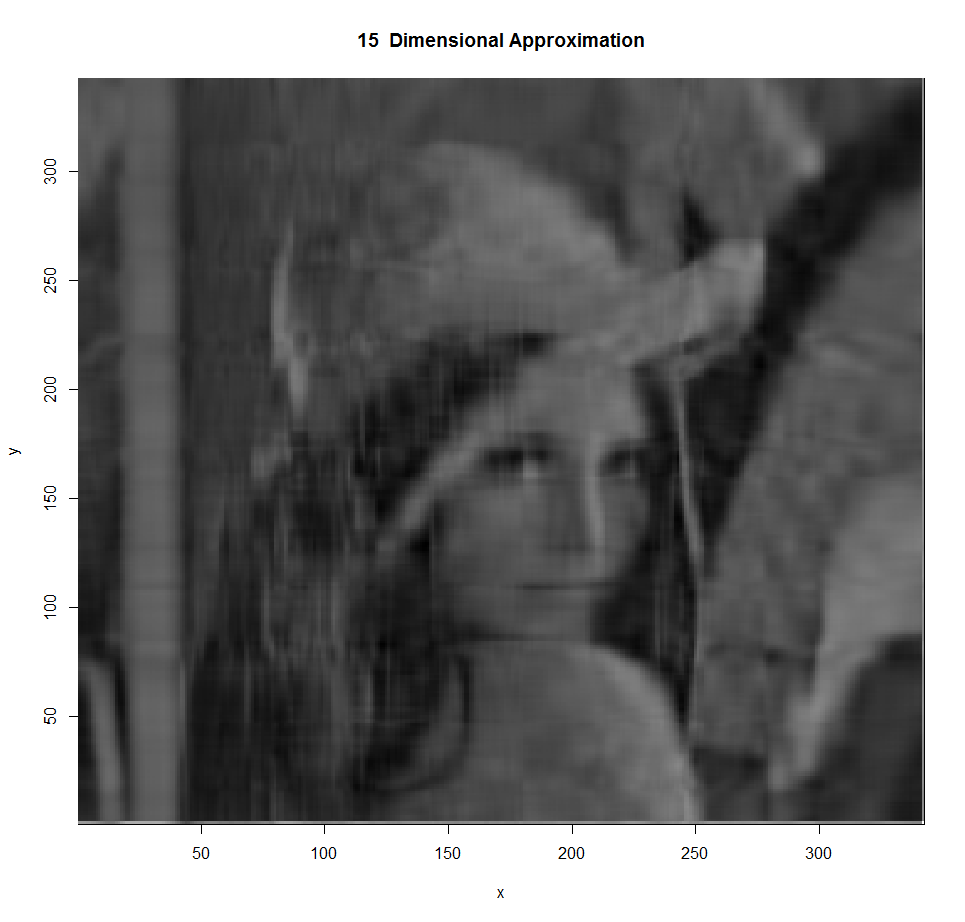
k = 10

> image.svd(Xsvd,k=10)



k = 15

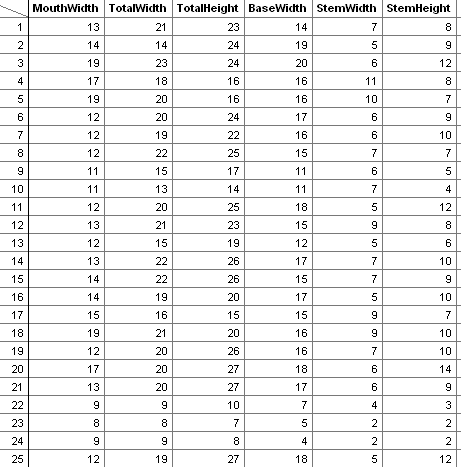
> image.svd(Xsvd,k=15)



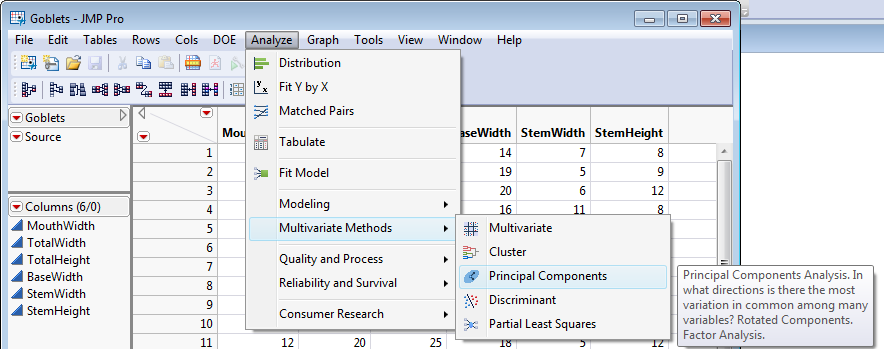
Etc…

**2.5 – Performing PCA in JMP**

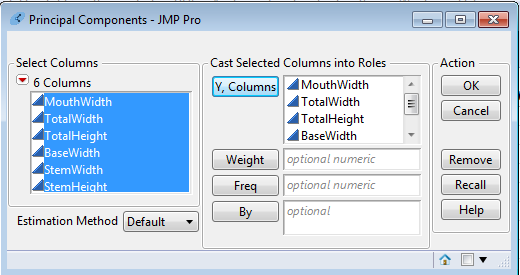
Example 2.1: Prehistoric Goblets  
Again using the goblet data in JMP (**Goblets.JMP**).



Select **Analyze > Multivariate Methods > Principal Components.**

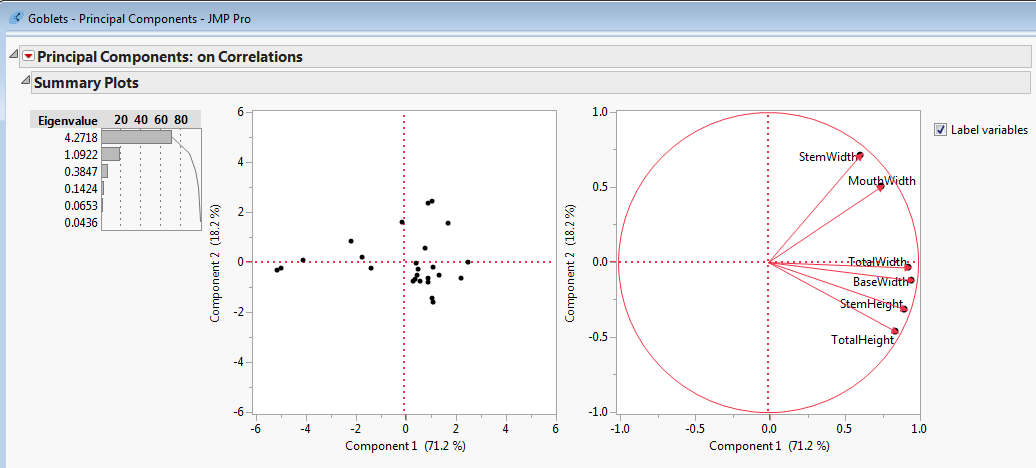


Place all variables to be included in your principal component analysis into the , Columns box. Click OK.



The default output is the eigenvalues, the percentage of total variation due to each principal component and the cumulative percentages.

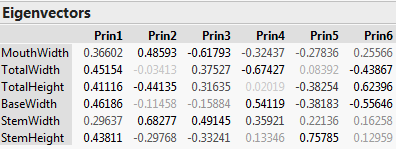
**Comment**: The default PC analysis in JMP uses the correlation matrix (i.e. the standardized variables).



The scree plot can be obtained from the drop-down menu.

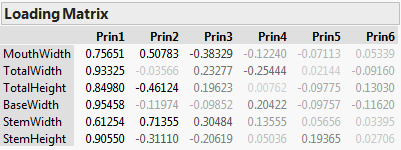


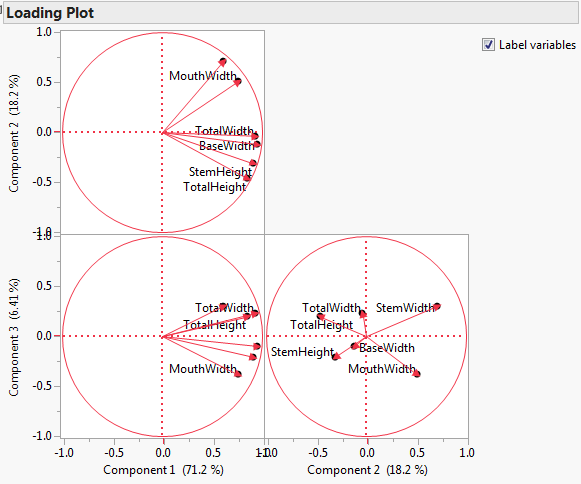
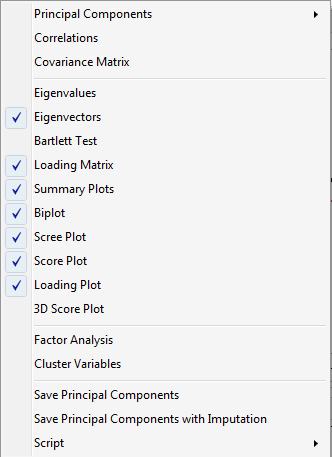
Likewise, the eigenvectors can be obtained from the red drop-down menu.

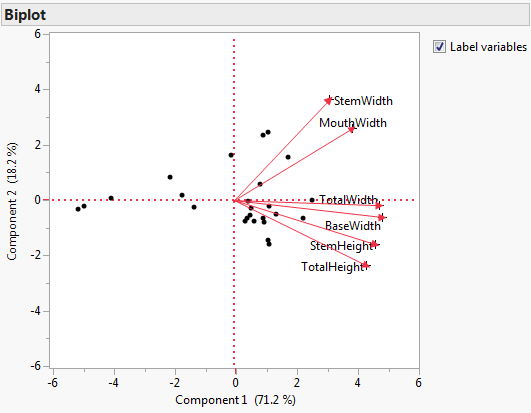


**Comment:**  The eigenvectors presented here are of the opposite sign of those computed by R. This has little effect on our interpretation of the principal components. For example, the second principal component is again contrasting the goblet’s width (stem and mouth) with its total height.

If you select Loading Matrix from the Principal Components drop-down menu you obtain the following:



These values are different than the eigenvectors that define the principal components but certainly share some similarities. These the pairwise correlations of each of the original with the principal component scores for each PC. These correlations give an alternative way to interpret what features of these data the different principal components are measuring.  
 



The Loading Plot shows the loading matrix values for two principal components at a time. The biplots combine the loading plots with a plot of the principal component scores () on the same graph.

**2.6 - Additional Examples of PCA**

Example 2.3: Weekly Stock Market Returns

The weekly rates of return for five stocks (JP Morgan, Citibank, Wells Fargo, Royal Dutch Shell, and ExxonMobil) listed on the New York Stock Exchange were determined over 103 successive weeks. The weekly rate of return was computed as



and is adjusted for stock splits and dividends.

The data in R…

JPMorgan\_Bank Citibank\_Bank WellsFargo\_Bank RoyalDutchShell\_Oil ExxonMobil\_Oil

1 0.0130338 -0.0078431 -0.0031889 -0.0447693 0.0052151

2 0.0084862 0.0166886 -0.0062100 0.0119560 0.0134890

3 -0.0179153 -0.0086393 0.0100360 0.0000000 -0.0061428

4 0.0215589 -0.0034858 0.0174353 -0.0285917 -0.0069534

5 0.0108225 0.0037167 -0.0101345 0.0291900 0.0409751

6 0.0101713 -0.0121978 -0.0083768 0.0137083 0.0029895

7 0.0111288 0.0280044 0.0080721 0.0305433 0.0032290

8 0.0484801 -0.0051480 0.0182495 0.0063348 0.0076752

9 -0.0344914 -0.0137991 -0.0080468 -0.0299011 -0.0108108

10 -0.0046596 0.0209882 -0.0060841 -0.0203940 -0.0126677

:

:

:

94 0.0373241 0.0359281 0.0252751 0.0581879 0.0169708

95 0.0238029 0.0031125 -0.0068757 0.0122545 0.0281715

96 0.0256826 0.0525266 0.0406957 -0.0316623 -0.0188482

97 -0.0060622 0.0086334 0.0058413 0.0445584 0.0305941

98 0.0217449 0.0229645 0.0291983 0.0084395 0.0319296

99 0.0033740 -0.0153061 -0.0238245 -0.0016738 -0.0172270

100 0.0033626 0.0029016 -0.0030507 -0.0012193 -0.0097005

101 0.0170147 0.0095061 0.0181994 -0.0161758 -0.0075614

102 0.0103929 -0.0026612 0.0044290 -0.0024818 -0.0164502

103 -0.0127948 -0.0143678 -0.0187402 -0.0049759 -0.0163732

The principal component analysis could be done on either the variance/covariance matrix or the correlation matrix.

PC Analysis based on variance/covariance matrix

> stock.pca=princomp(Stock)

> summary(stock.pca)

Importance of components:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5

Standard deviation 0.03680217 0.02635056 0.01585365 0.01188352 0.01085046

Proportion of Variance 0.52926066 0.27133298 0.09821584 0.05518400 0.04600652

Cumulative Proportion 0.52926066 0.80059364 0.89880948 0.95399348 1.00000000

PC Analysis based on correlation matrix

> stock.pca.cor=princomp(Stock,cor=TRUE)

> summary(stock.pca.cor)

Importance of components:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5

Standard deviation 1.5611768 1.1861756 0.7074693 0.63248050 0.50514343

Proportion of Variance 0.4874546 0.2814025 0.1001025 0.08000632 0.05103398

Cumulative Proportion 0.4874546 0.7688572 0.8689597 0.94896602 1.00000000

The outcomes are similar in this case because the variables (i.e. weekly rate of return) are similar for these stocks. We will continue with the analysis based on the correlation matrix.

> stock.pca.cor$loadings

Loadings:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5

JPMorgan\_Bank -0.469 0.368 0.604 0.363 0.384

Citibank\_Bank -0.532 0.236 0.136 -0.629 -0.496

WellsFargo\_Bank -0.465 0.315 -0.772 0.289

RoyalDutchShell\_Oil -0.387 -0.585 -0.381 0.595

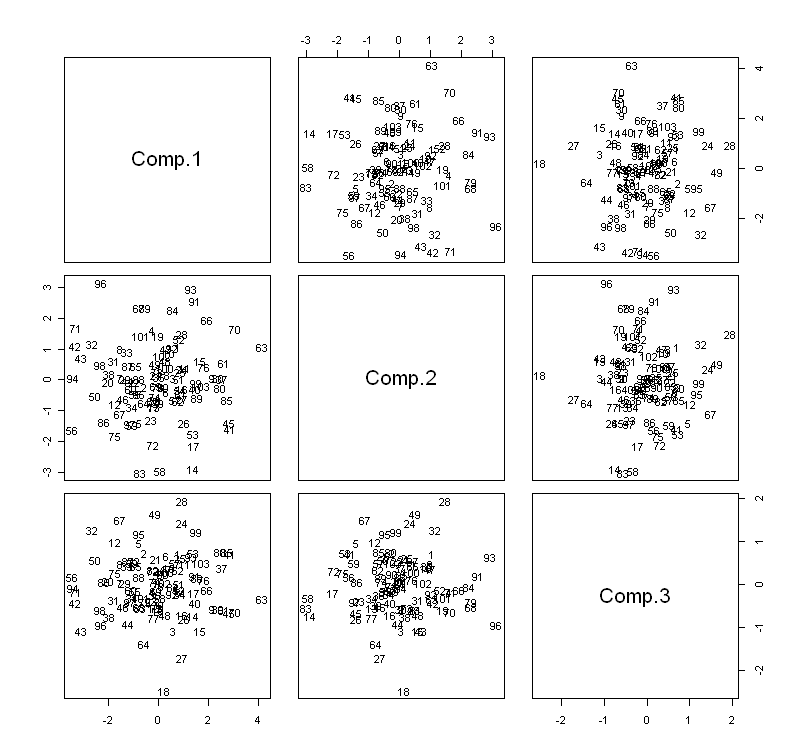
ExxonMobil\_Oil -0.361 -0.606 0.109 0.493 -0.498

**Questions:**

1. What is the make-up of the 1st principal component?
2. What is the make-up of the 2nd principal component?
3. In this analysis it appears we gain some additional information by considering the third principal component. What is the make-up of the 3rd principal component?

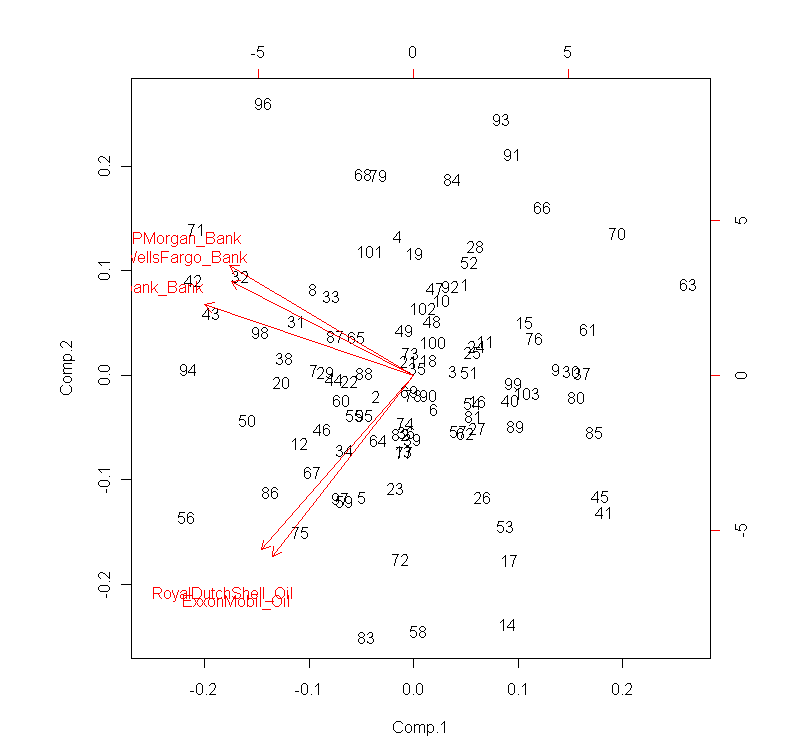
Plotting the first three principal components in a pairwise fashion.

> pairs(stock.pca.cor$scores[,1:3],panel=function(x,y){text(x,y,1:103)})



Getting the biplot for this example…

> biplot(stock.pca.cor)



Understanding the outlying points in this plot…

> outlying=c(96,93,91,63,45,41,14,58,83,56,86,94,42,71)

> Stock[outlying,]

JPMorgan\_Bank Citibank\_Bank WellsFargo\_Bank RoyalDutchShell\_Oil ExxonMobil\_Oil

96 0.0256826 0.0525266 0.0406957 -0.0316623 -0.0188482

93 0.0307783 -0.0160888 0.0031045 -0.0539478 -0.0556609

91 0.0090063 -0.0022422 0.0000000 -0.0429774 -0.0620229

63 -0.0300058 -0.0497446 -0.0167189 -0.0507510 -0.0583157

45 -0.0446763 -0.0408118 -0.0163225 -0.0035049 -0.0008137

41 -0.0329997 -0.0313480 -0.0362141 0.0055866 -0.0065208

14 -0.0458668 -0.0278243 -0.0142696 0.0374776 0.0332022

58 -0.0260532 -0.0168492 -0.0080604 0.0432676 0.0587486

83 -0.0265896 -0.0002339 -0.0033698 0.0615551 0.0561091

56 0.0188149 0.0379692 0.0154985 0.0510400 0.0784157

86 0.0136240 0.0182335 0.0086520 0.0568640 0.0387476

94 0.0373241 0.0359281 0.0252751 0.0581879 0.0169708

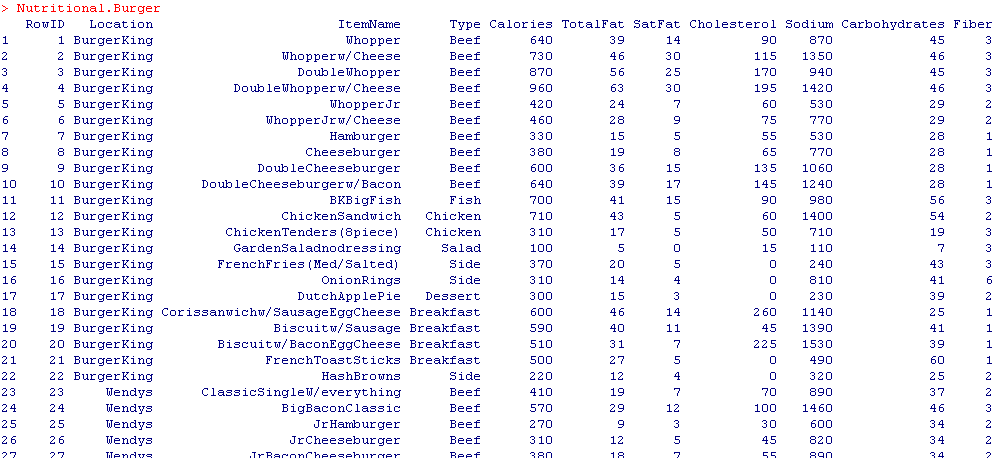
42 0.0302098 0.0522778 0.0317662 0.0267857 0.0105865

71 0.0465672 0.0410867 0.0349723 0.0152170 0.0056721

Example 2.4: Nutritional Information for Three Fast Food Chains

For this example, consider Burger King, Wendy’s and McDonald’s from the Nutritional dataset.

> Nutritional.Burger=Nutritional[1:65,]





Doing an **incorrect** PC analysis (i.e. using the covariance matrix instead of the correlation matrix).

> Nutritional.pca=princomp(Nutritional.Burger[,5:11])

> Nutritional.pca

Call:

princomp(x = Nutritional.Burger[, 5:11])

Standard deviations:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7

398.629915 135.424524 35.591907 9.003670 2.744263 2.416475 1.496017

7 variables and 65 observations.

> summary(Nutritional.pca)

Importance of components:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7

Standard deviation 398.629915 135.4245239 35.591906870 9.0036698100 2.7442630184 2.416475e+00 1.496017e+00

Proportion of Variance 0.889685 0.1026812 0.007092494 0.0004538743 0.0000421646 3.269347e-05 1.253053e-05

Cumulative Proportion 0.889685 0.9923662 0.999458737 0.9999126114 0.9999547760 9.999875e-01 1.000000e+00

> Nutritional.pca$loadings

Loadings:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7

Calories -0.322 0.937

TotalFat 0.256 -0.951 0.117

SatFat 0.184 0.957 -0.209

Cholesterol -0.969 -0.220

Sodium -0.941 -0.329

Carbohydrates 0.212 -0.920 0.154 -0.276

Fiber 0.226 0.969

**Questions:**

1. What is the make-up of the 1st principal component?
2. What is the make-up of the 2nd principal component?
3. Why is Fiber absent from the first four principal components?   
   **Hint**: Compare the scale for Fiber to Sodium or Calories.

Doing a **correct** PC analysis (i.e. using the correlation matrix which again is equivalent to standardizing your data and running your PC analysis).

> Nutritional.pca.cor=princomp(Nutritional[,5:11],cor=TRUE)

> summary(Nutritional.pca.cor)

Importance of components:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7

Standard deviation 2.0117816 1.2518205 0.76936072 0.7031833 0.42906654 0.31955216 0.114393472

Proportion of Variance 0.5781808 0.2238649 0.08455942 0.0706381 0.02629973 0.01458766 0.001869409

Cumulative Proportion 0.5781808 0.8020457 0.88660511 0.9572432 0.98354294 0.99813059 1.000000000



**Question:**

1. How many principal components are necessary for this analysis?

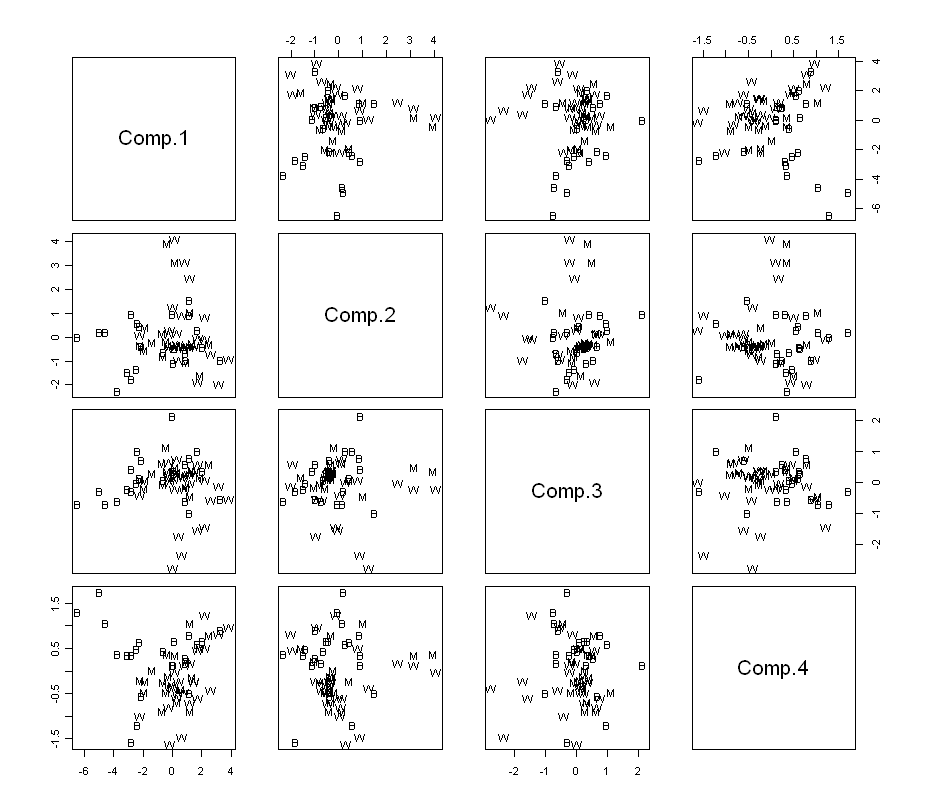
Plotting the first four principal components…

First, creating a label for the plot

> mylabel=c(rep("B",22),rep("W",25),rep("M",18))

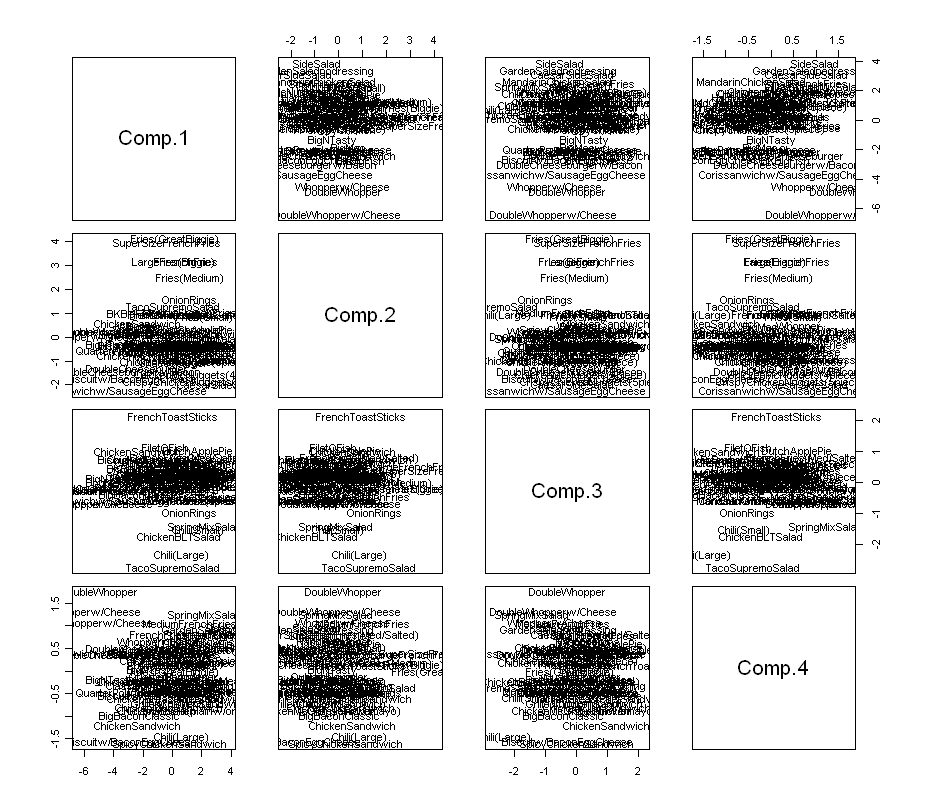
Using the pairs function to create the pairwise plots…

> pairs(Nutritional.pca.cor$scores[,1:4], panel=function(x,y){text(x,y,mylabel)})



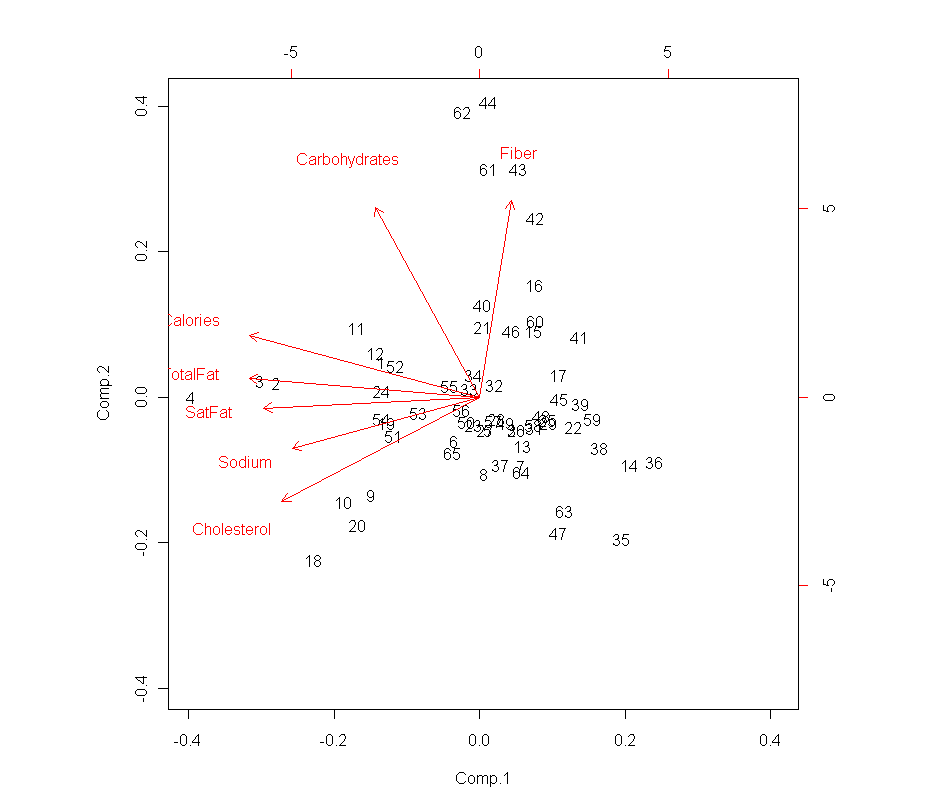
Using the pairs function to create the pairwise plots (using item as a label)

> pairs(Nutritional.pca.cor$scores[,1:4],   
 panel=function(x,y){text(x,y,Nutritional.Burger[,3])})

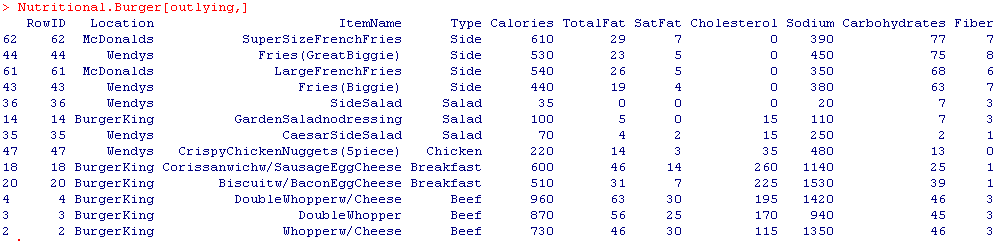


Obtaining the biplot for this analysis…

> biplot(Nutritional.pca.cor)



> outlying=c(62,44,61,43,36,14,35,47,18,20,4,3,2)



* 1. **– Independent Component Analysis (ICA)**

Independent Component Analysis (ICA) is a multivariate statistical technique that seeks to uncover the hidden variables in high-dimensional data. As such, it belongs to what is called the *latent variable* class of models.

A classic problem that Independent Component Analysis (ICA) can solve is the “Cocktail Party” problem where separate microphones record the ambient sound in at a cocktail party. Each microphone will pick up several different conversations and sounds in the room producing an audio stream with several different sound components combined. ICA will break apart the sound signals from each microphone into the independent underlying sound components. A great demo (with audio) of ICA used in this capacity can be found at the ICA Research center at the Helsinki University of Technology (<http://research.ics.aalto.fi/ica/cocktail/cocktail_en.cgi>).

Below is a simple example

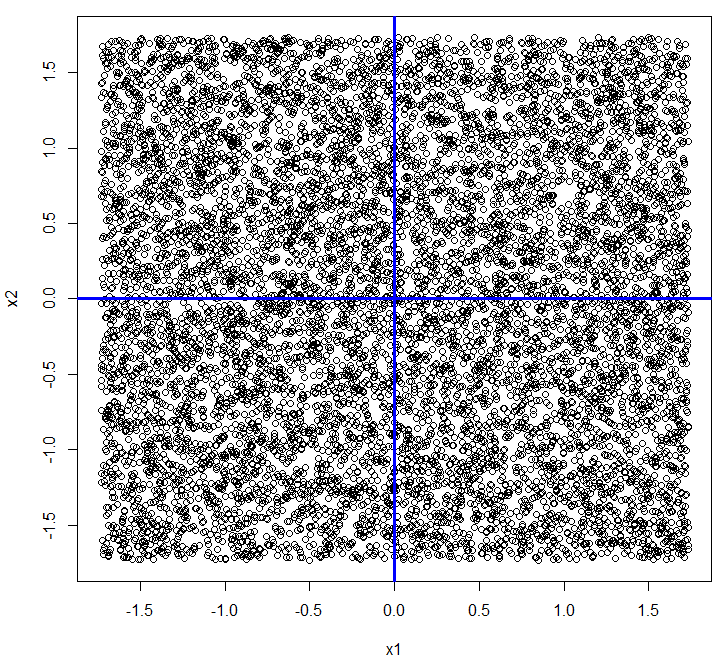
Let and independent.

> x1 = runif(10000,-sqrt(3),sqrt(3))

> x2 = runif(10000,-sqrt(3),sqrt(3))

> plot(x1,x2)

> abline(h=0,v=0,lwd=3,col=”blue”)



We can mix these two independent sources by multiplying the matrix by a *mixing matrix* , i.e. compute **.** Plotting the columns of we obtain.

> A = matrix(c(2,3,2,1),2,2,byrow=T)

> A  
 [,1] [,2]

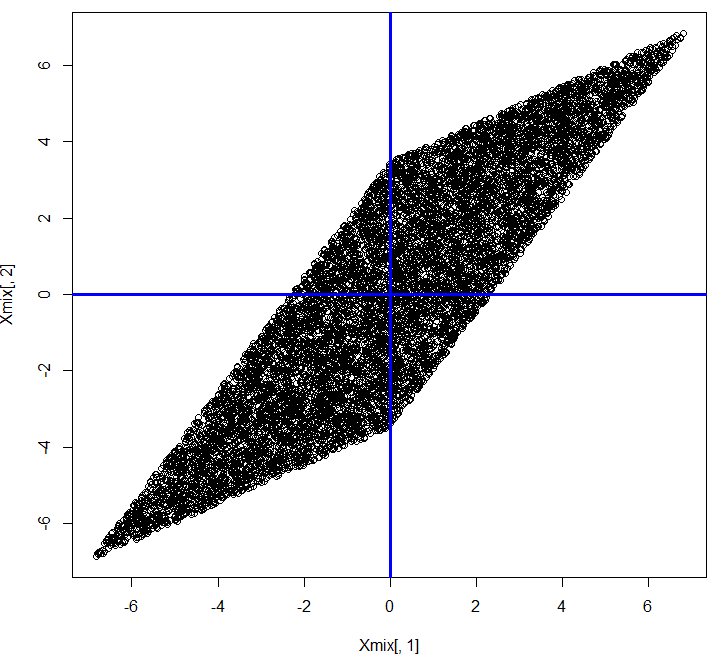
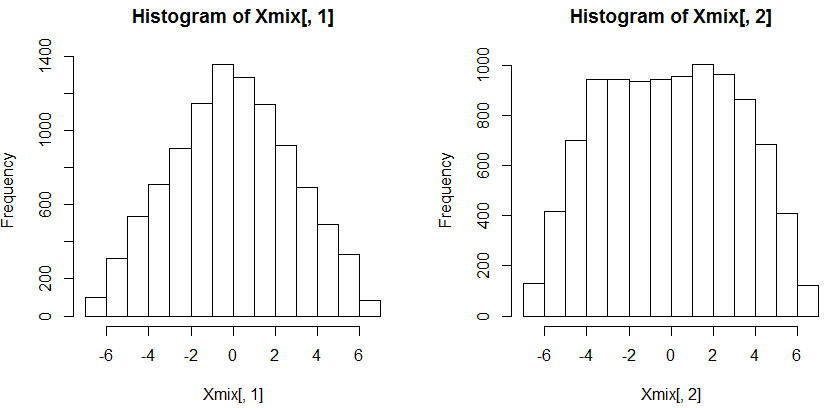
[1,] 2 3

[2,] 2 1

> X = cbind(x1,x2)

> Xmix = X%\*%A

> plot(Xmix[,1],Xmix[,2])

> abline(h=0,v=0,lwd=3,col=”blue”)  
> hist(Xmix[,1]); hist(Xmix[,2])  
 

Clearly the columns of are not INDEPENDENT! This would certainly be the case of separate microphones recording the sounds in the same room. Can independent component analysis recover the independent components that were mixed together to form the signals displayed above?

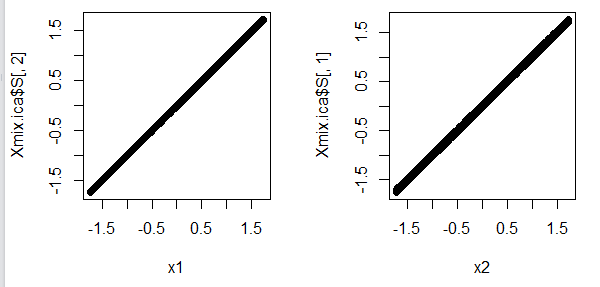
> Xmix.ica = fastICA(Xmix,n.comp=2)  
> attributes(Xmix.ica)

$names

[1] "X" "K" "W" "A" "S"

ICA seeks to find the independent signals used in creating the observed mixtures. These are stored in the matrix and the estimated mixing matrix is given by .

> plot(x1,Xmix.ica$S[,2])

> plot(x2,Xmix.ica$S[,1])  


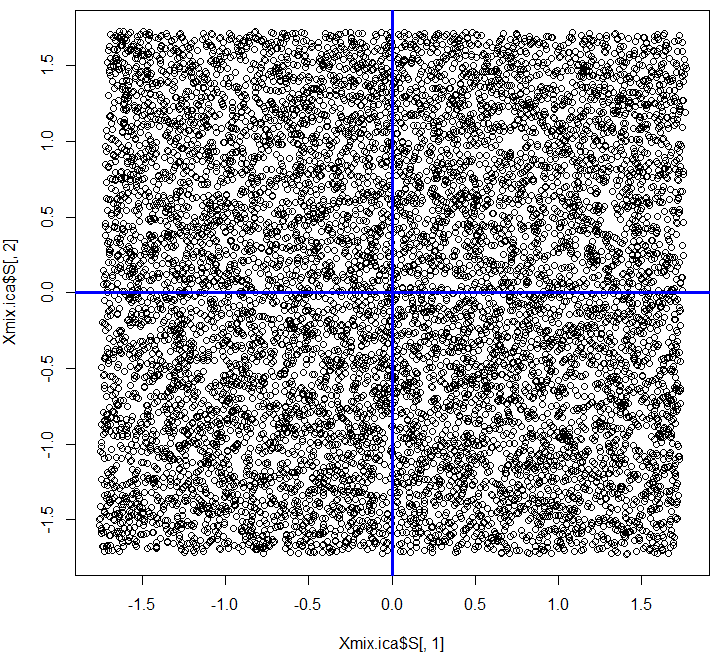
> Xmix.ica$A

[,1] [,2]

[1,] 1.996025 1.003487

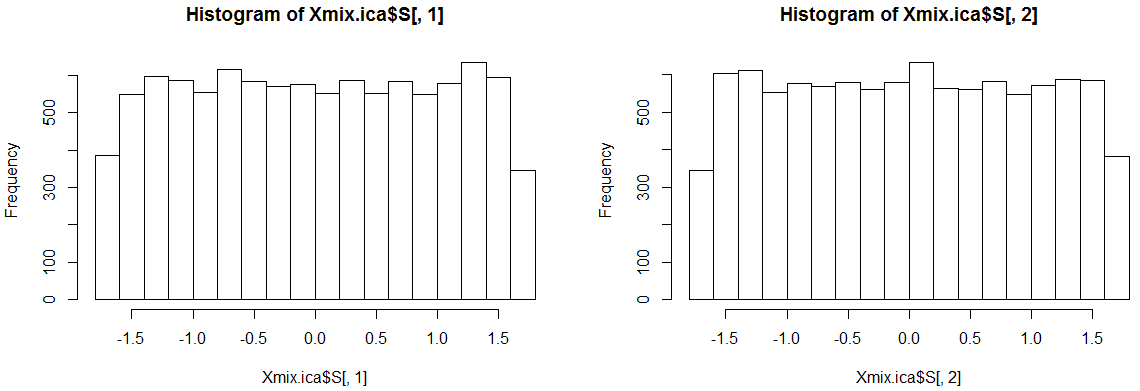
[2,] 1.980538 3.002365

> plot(Xmix.ica$S[,1],Xmix.ica$S[,2])



> hist(Xmix.ica$S[,1])

> hist(Xmix.ica$S[,2])



We can see that ICA has successfully estimated the mixing matrix with a very small error and has found the two independent sources that were mixed in order to create observed data (the parallelogram).

* 1. **– Some of the Mathematical Details of ICA**

The general ICA model assume that , the data matrix containing the mixed sources, is generated by

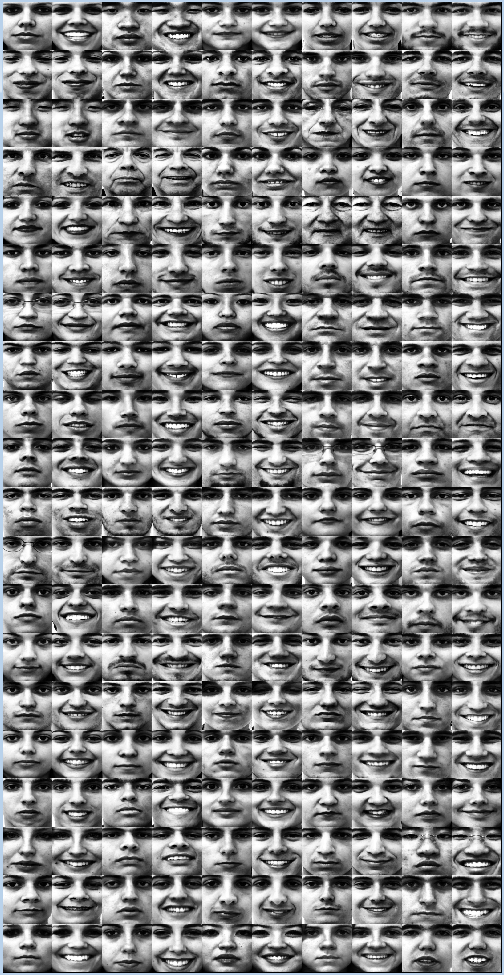
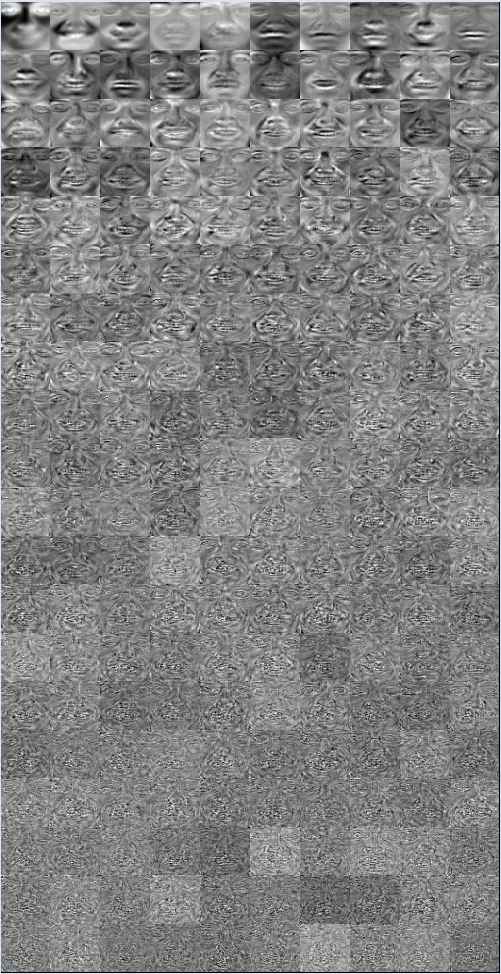
where is an unobservable random matrix whose columns are the independent components that when mixed by the function produce the observed data matrix with zero-mean additive error given by . In the cocktail party problem might represent the static or distortion in the recording process. If we assume that is linear function, i.e. each observed signal in is a linear combination of the independent sources contained in , then it can be represented using a mixing matrix , as in the example above.

The ICA model then becomes,

Which means that the observed value of the source in can be represented as,

or the source in can be estimated as,

where is the column of **.**

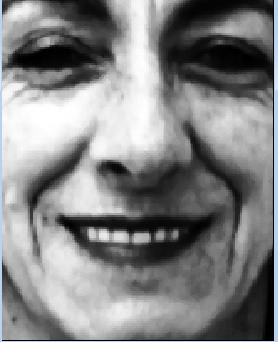
Example 2.5 - Brazilian Face Data  
The FEI face database is a Brazilian face database that contains a set of face images taken between June 2005 and March 2006 at the Artificial Intelligence Laboratory of FEI in São Bernardo do Campo, São Paulo, Brazil. There are 14 images for each of 200 individuals, a total of 2800 images. All images are colorful and taken against a white homogenous background in an upright frontal position with profile rotation of up to about 180 degrees. Scale might vary about 10% and the original size of each image is 640x480 pixels. All faces are mainly represented by students and staff at FEI, between 19 and 40 years old with distinct appearance, hairstyle, and adorns.  The number of male and female subjects are exactly the same and equal to 100. Figure 1 shows some examples of image variations from the FEI face database.  
 

We can reconstruct any of the 200 faces by taking the full linear combination of the 200 *eigenfaces* obtained from the singular value decomposition (SVD), i.e. finding the 200 principal components, of the pixel-image matrix .

Original 200 Facial Images

Eigenfaces from SVD

Reconstruction of the 28th Face using the 200 Eigenfaces



The linear combination partially shown below will produce the image on the left perfectly, i.e. without error.



Doing this in R

> dim(Ximages)

[1] 31266 200

This matrix contains the images in columns and the 31,266 pixels that make up each images as rows. If we take the transpose of this matrix, then the rows will correspond to the 200 images and the 31266 columns will contain the pixels. We will conduct a singular value decomposition of the transposed image matrix.

> tX = t(Ximages)  
> dim(tX)

[1] 200 31266

> faces.svd = svd(tX)

> attributes(faces.svd)

$names

[1] "d" "u" "v"

> dim(faces.svd$u)

[1] 200 200

> dim(faces.svd$v)

[1] 31266 200

> dim(diag(faces.svd$d))

[1] 200 200

The eigenfaces displayed on the previous page are the columns of the matrix .

> brazil.plot

function(X,i=1){

pixels = X[,i]

pixmat = matrix(pixels,193,162)

pixmat = apply(pixmat,2,rev)

image(1:162,1:193,t(pixmat),col=gray(c(0:192)/192))

}

> brazil.plot(faces.svd$v,1); brazil.plot(faces.svd$v,2)  


As shown above we reconstructed image 28 (the 28th column of the Ximages matrix or the 28th row of the tX matrix) using a linear combination of the 200 *eigenfaces*.

Recall the SVD of a matrix is given by . Thus the transpose of the matrix X is given by We will use this face to find the mixing coefficients of the eigenfaces to reproduce the original image.

The matrix of linear combinations of eigenfaces needed to produce the original images are the columns of the matrix .

> mix.mat = diag(faces.svd$d)%\*%t(faces.svd$u) 🡨 Form

> face28 = faces.svd$v%\*%mix.mat[,28]

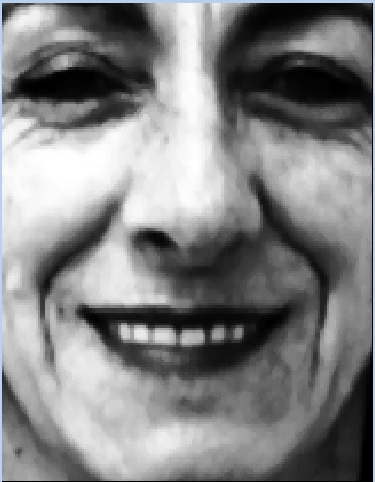
Check it…

> brazil.plot(face28)



Independent Components

Facial Image for the 28th Subject



S = face.ICAfull$S  
A = face.ICAfull$A

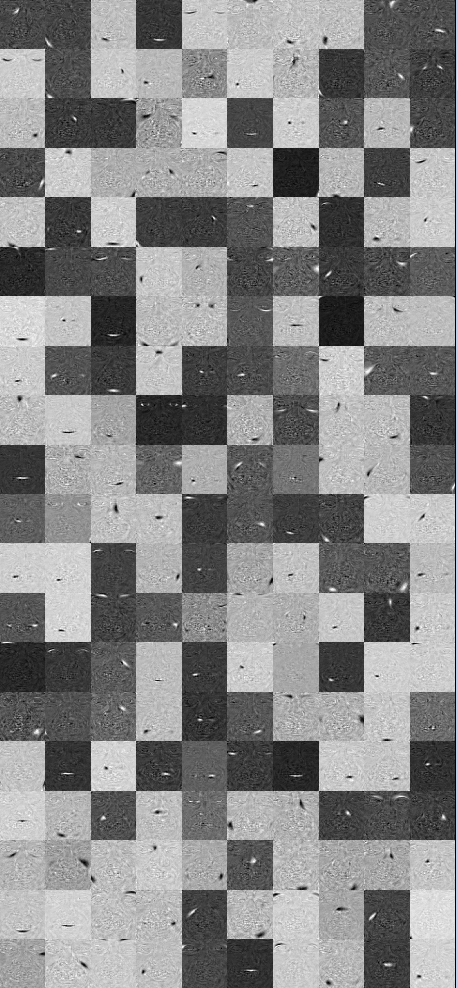
To obtain the reconstruction of the 28th facial image we can take the linear combination of the 200 basis images specified by the 28th column of the mixing matrix .

Face28 = S %\*% A[,28]

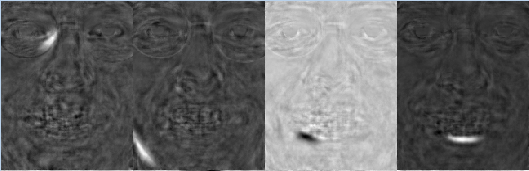
 

Each of the 200 images in original set can be perfectly reconstructed by taking the appropriate linear combination of 200 independent components, i.e. basis images, shown on the left.



The URL below will show a movie animating this reconstruction.

<https://mediaspace.mnscu.edu/media/Face+28+ICA+2/0_zmbc0iid>

Larger Versions of First Four Independent Components  


We can animate the process of linearly combining the basis images (from PCA or ICA) using the function Progression4 below.

> face.ICAfull = fastICA(Ximages,n.comp=200)

> tX = t(Ximages)

> face.svd = svd(tX)

Progression4 = function(d=.1,k=28) {  
 for (i in 1:200) {

ord = order(abs(face.ICAfull$A[,k]),decreasing=T)

temp = faces.svd$v[,c(1:i)]%\*%matrix(mix.mat[c(1:i),k])

par(mfrow=c(2,2),mar=c(0,0,0,0),xaxt="n",yaxt="n")

brazil.plot(temp)

brazil.plot(as.matrix(faces.svd$v[,i]))

temp = face.ICAfull$S[,ord[c(1:i)]]%\*%matrix(face.ICAfull$A[ord[c(1:i)],k])

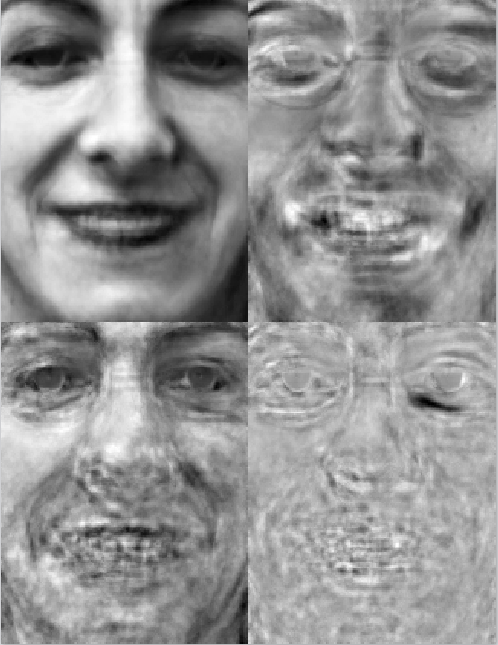
brazil.plot(temp)

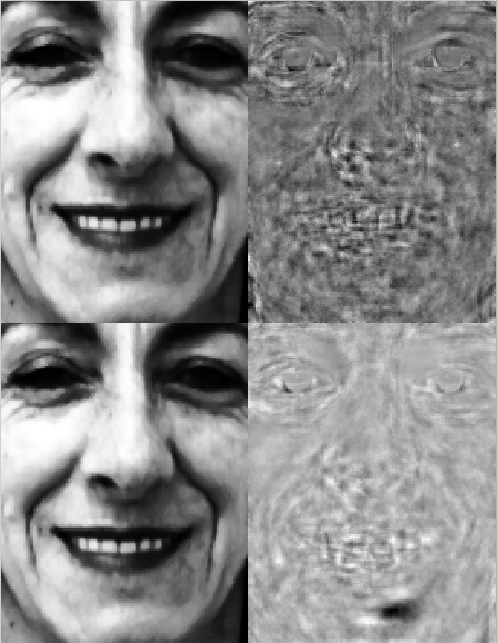
brazil.plot(as.matrix(face.ICAfull$S[,ord[i]]))

Sys.sleep(d)

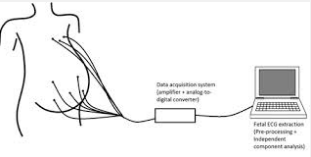
}

}

Example 2.6 – ECG readings for Pregnant Woman   
(de Lanthauwer, de Moor, and Vandewalle, 2000)

  
  
In prenatal diagnostics, it is important for a physician to be able to monitor – in a non-invasive way – the fetal heart activity of a pregnant woman so that the health and condition of the fetus can be assessed. A multi-channel electrocardiogram (ECG) can be used to obtain a mixture of maternal and fetal electrical activity, including fetal heart rate and maternal heart rate; however, the maternal ECG signal is many hundreds or thousands of times stronger than the fetal ECG signal, and these signals are further contaminated by respiration baseline wandering and other sources of electrical interference.

The data for this example consist of 2,500 ECG points sampled at 500 mHz using 8-channel cutaneous (i.e., on the skin) potential recordings of a pregnant woman. The 8 sets of cardiac rhythms are displayed below. The 2,500 points are recorded over a period of 5 seconds, one point every 0.002 seconds. Note that the range of amplitudes increases as we go from Channel 1 to Channel 8. The first five channels (1-5) are measured near the fetus and hence the mother’s abdomen. Fetal contributions are somewhat visible in Channels 1,2, and 3, but their magnitudes are quite weak (see plots below). The other three channels (6-8) were placed on the mother’s chest, near the heart; note that the high magnitudes of the mother’s ECG signals tend to drown out the fetal ECG signals.

> fmECG = read.table(file.choose(),header=T,sep=”,”)

> names(fmECG)

[1] "Frequency" "Channel.1" "Channel.2" "Channel.3" "Channel.4" "Channel.5" "Channel.6"

[8] "Channel.7" "Channel.8"

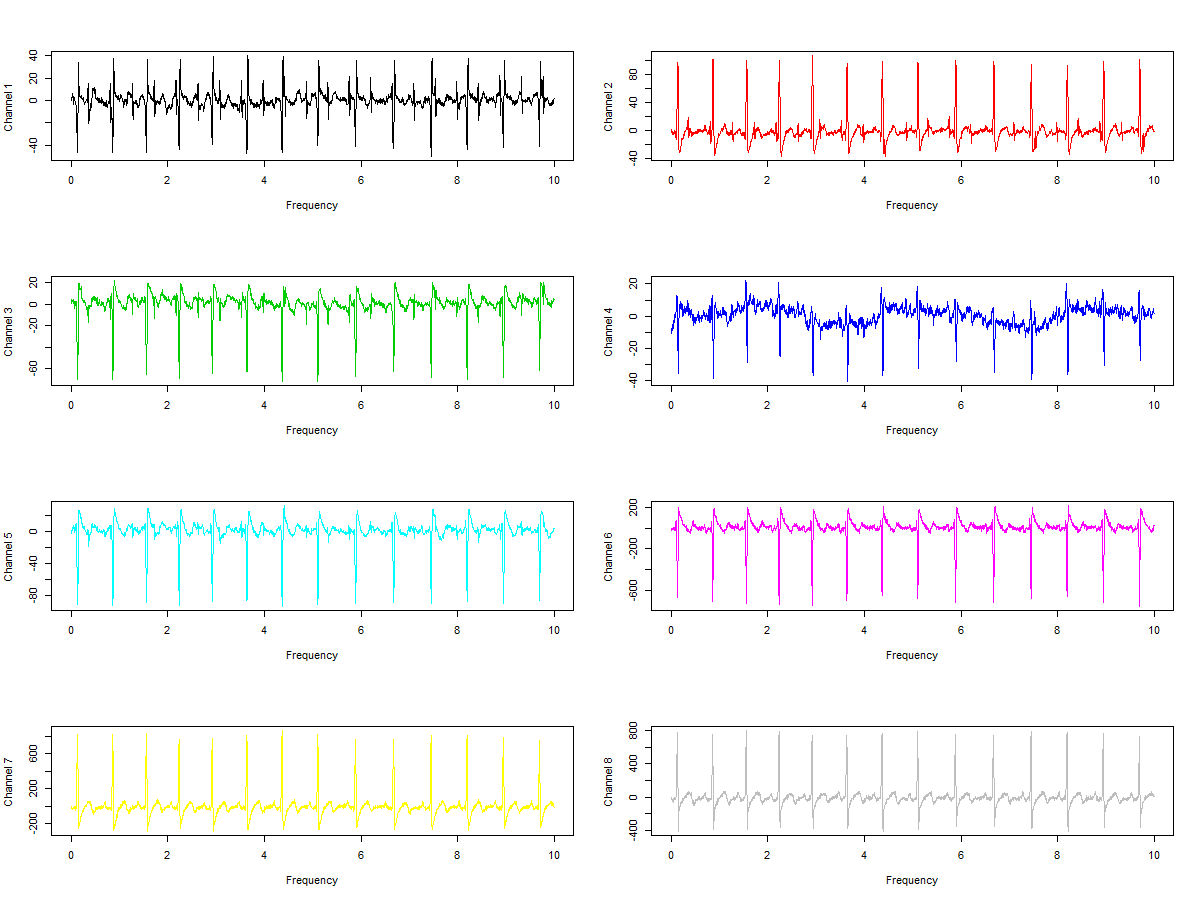
Plot the original channel or sensor readings from the eight ECG sensors.

> par(mfrow=c(4,2)) 🡨 sets up a 4 rows and 2 columns of plots

> for (i in 1:8){

plot(1:nrow(fmECG),fmECG[,i+1],xlab="",ylab=paste("Sensor ",as.character(i)),  
type="l",col=i+1)   
}

Plots of the original ECG recordings from the 8 sensors (channels)



We can see the fetal contributions in sensors 1 – 3 (higher frequencies) but there magnitudes are very weak. Sensors 4 is clearly picking up a low frequency signal which could be related to mothers respiration? Sensors 6 – 8 are placed near the mother’s chest, so the maternal ECG readings dominate and are much stronger as evidenced by the magnitudes (*see y-axis ranges*).

Next we will perform an independent component analysis of the scaled sensor signals.

> library(ica)

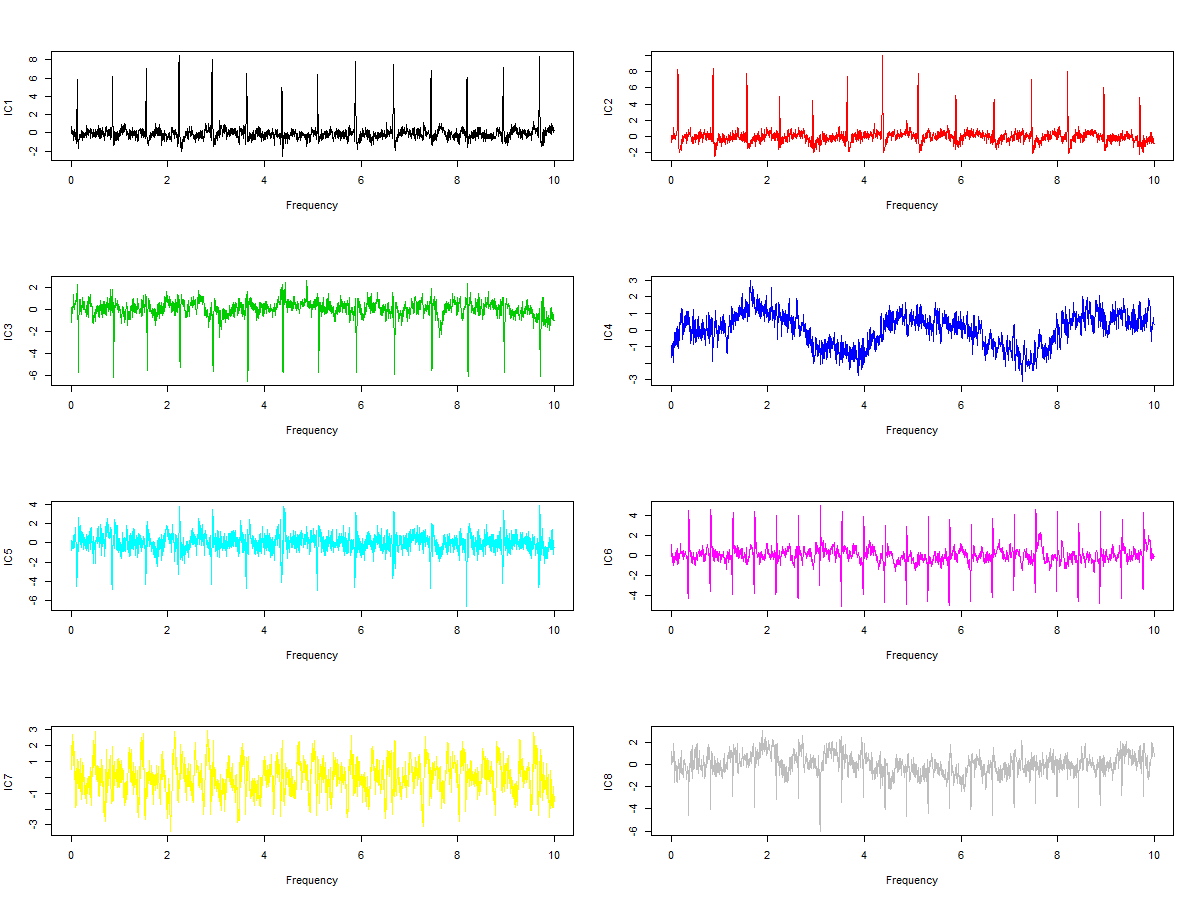
> ECG.mat = scale(fmECG[,-1]) 🡨 create a matrix containing the eight scaled sensor readings.  
> fmECG.ica = icafast(ECG.mat,nc=8)

> par(mfrow=c(4,2)) 🡨 sets up a 4 rows and 2 columns of plots

> for (i in 1:8){

plot(1:nrow(fmECG),fmECG.ica[,i],xlab="",ylab=paste("Sensor ",as.character(i)),  
type="l",col=i+1)   
}

Plots of the Independent Components from the Maternal-Fetal ECG Sensors

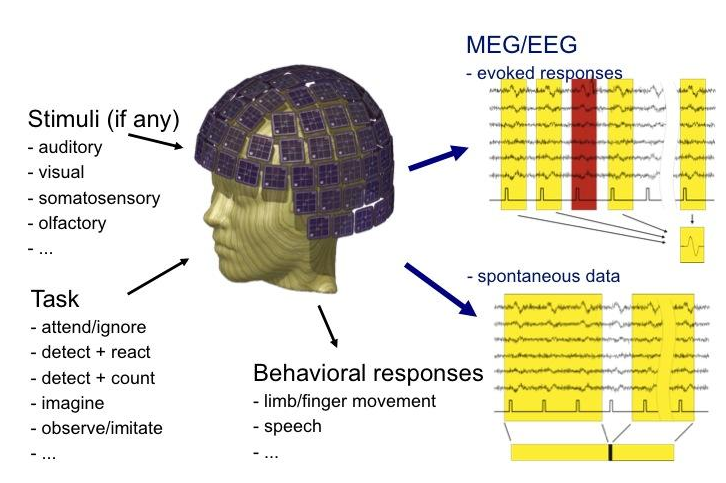


The eight independent components are shown above. Independent components IC6 and IC8 show give the fetal cardiac rhythms, independent components IC1, IC2, IC3, and IC5 reflect the cardiac rhythms of the mother, IC4 shows a respiration component, and IC7 shows noise level of the sensors. These conclusions would be reached in consultation with the physician with the contextual knowledge of the ECG monitoring of a mother carrying a fetus.

Example 2.8 – Magnetoencephalography (MEG)



Magnetoencephalography (MEG) is a functional   
neuroimaging technique for mapping brain activity by   
recording magnetic fields produced by electrical currents   
occurring naturally in the brain, using very sensitive   
magnetometers. Arrays of SQUIDs (superconducting   
quantum interference devices) are currently the most   
common magnetometer, while the SERF   
(spin exchange relaxation-free) magnetometer is being   
investigated for future machines. Applications of MEG   
include basic research into perceptual and cognitive brain   
processes, localizing regions affected by pathology before   
surgical removal, determining the function of various parts   
of the brain, and neurofeedback. This can be applied in a   
clinical setting to find locations of abnormalities as well as   
in an experimental setting to simply measure brain activity.



In a noninvasive experiment carried out by the ICA group at the Helsinki University of Technology (Jousmäki, et al. 1997), the MEG signals of a test subject were taken using a whole-scalp neuromagnetometer (see picture above) with 122 SQUID sensors organized in pairs at 61 grid locations uniformly distributed around the head. The weak magnetic fields produced by brain activity are detected by these sensors. The MEG signals are stimulated by having the test perform the following tasks: (1) blink their eyes, (2) make quick, simultaneous movements with both eyes at the same time in the same direction to simulate ocular artifacts (called *saccades*), (3) bite his teeth for as long as 20 seconds to simulate muscle artifacts, (4) a piece of metal was placed near the navel to capture breathing artifacts; and (5) a digital watch was placed one meter away from the helmet in the shielded room to simulate a general artifact.

The resulting MEG data consisted of n = 17,730 amplitude measurements for the 122 MEG sensors over a period of 2 minutes. During that time the subject was asked to perform activities (2), (1), and (3) in that order. The watch and metal near the diaphragm were present throughout. A sample of 40 of the original signals from the 122 sensors is shown below.

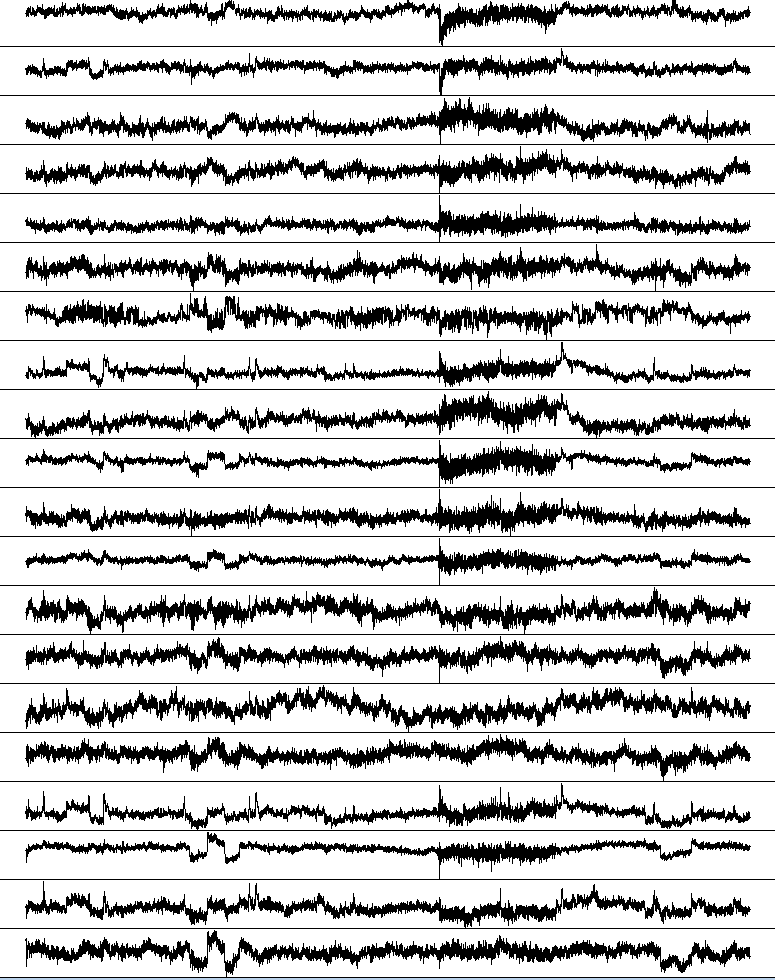
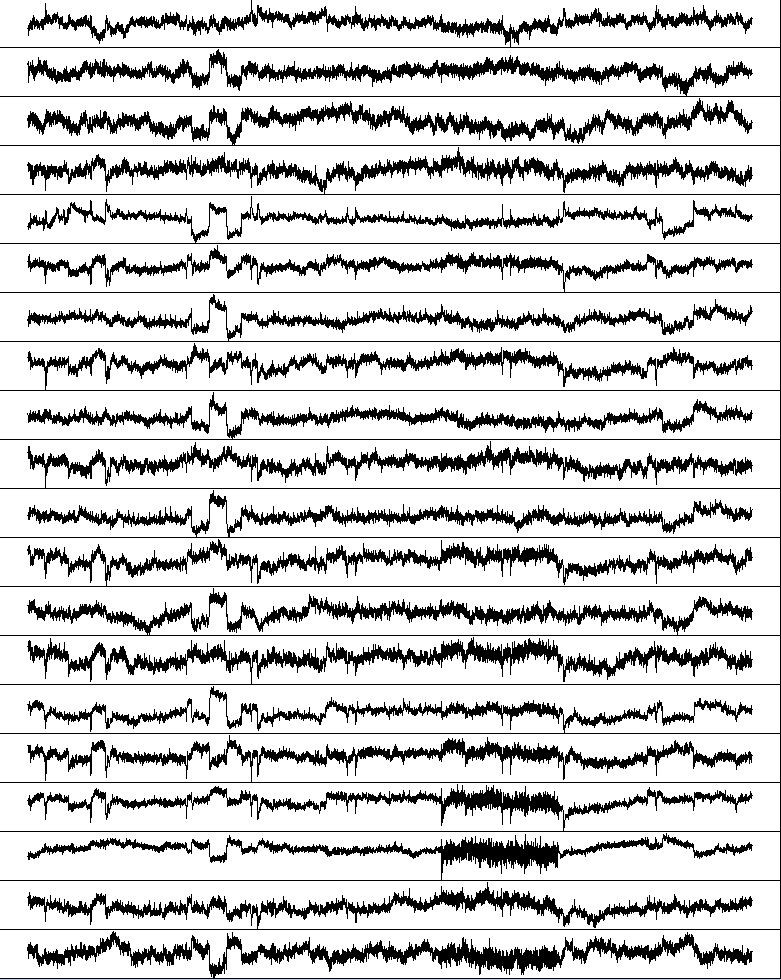
> EEGMEG = read.table(file.choose(),header=T,sep=”,”) 🡨 Read in **EEGMEG.csv**

> ind = sample(1:122,40,replace=F)

> par(mfrow=c(20,1),mar=c(0,0,0,0),ask=T)

> for (i in ind) {plot(1:nrow(EEGMEG),EEGMEG[,i],xaxt=”n”,yaxt=”n”,type=”l”)}

*40 randomly selected MEG sensor readouts*

As we can see many of the sensors yield similar patterns over the two minute recording period, thus we might expect to have reasonable correlation structure amongst 122 sensor readouts.

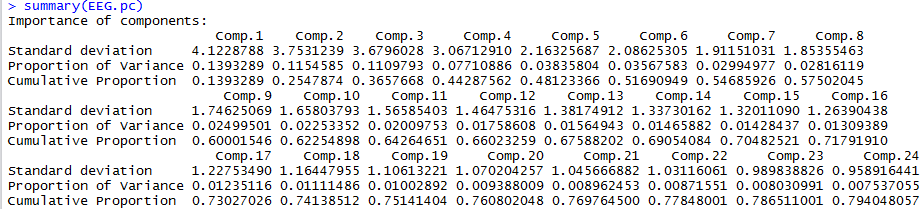
> EEG.cor = cor(EEGMEG)

> library(corrplot)

> corrplot(EEG.cor,order=”hclust”,tl.cex=0.5)



> EEG.pc = princomp(EEGMEG,cor=T)  
> summary(EEG.pc)



> EEG.scores = EEG.pc$scores[,1:22]

We can now use ICA to extract the underlying structure in contained in first 22 principal components, which account for 77.85% of the total variation in the original 122 EEG-MEG readings.

> EEG.ica = fastICA(EEG.scores,n.comp=22)

> par(mfrow=c(22,1),mar=c(0,0,0,0),xaxt="n",yaxt="n")

> for (i in 1:22){plot(1:nrow(EEGMEG),EEG.ica$S[,i],type="l")}

Independent Component Analysis of the MEG – 22 Principal Components

IC1

IC2

IC3

IC4

IC5

IC6

IC7

IC8

IC9

IC10

IC11

IC12

IC13

IC14

IC15

IC16

IC17

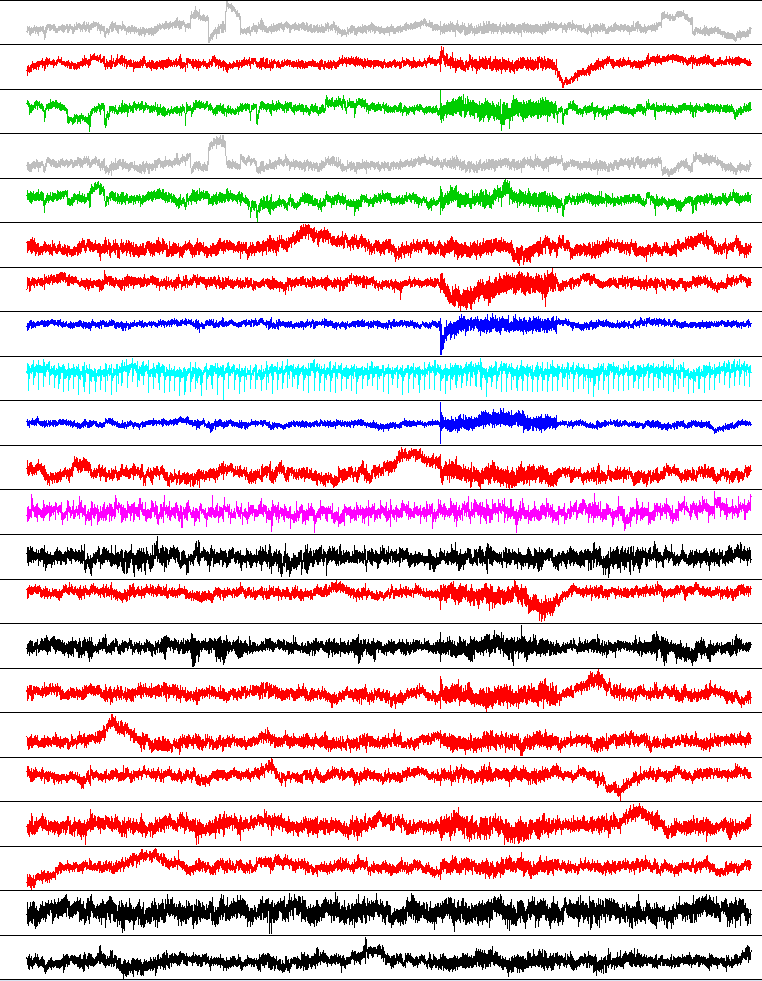
IC18

IC19

IC20

IC21

IC22



Examination of the independent components leads to the following interpretations:

IC1 & IC4 – Saccades, i.e. horizontal eye movements.

IC3 & IC5 – blinking eyes

IC12 – digital watch

IC8 & IC10 – biting down

IC9 – cardiac cycle

IC2, IC6, IC7, IC11, IC14, IC16-IC20 – overlearning or breathing (large rolling peaks)

IC13, IC15, IC21-IC22 – error or noise components

Note these conclusions would need to be made through discussions with the neuroscientists who conducted experiment; utilizing their contextual knowledge of the EEG-MEG process and the electrical activities in the brain.

**Questions and Tasks**

Using the data in **World Bank.csv** conduct a thorough investigation of these data.

* Examine appropriate plots of the variables in these data.
* Examine correlations amongst these variables (use corrplot)
* Conduct a thorough PCA of these data using the entire dataset or a subset of these variables of your choosing (you must use at least 15 variables).

Be prepared to present your results Friday morning!