R is an object oriented language. We will use R for statistical analysis in FIN 504/ORF 504. To download R, go to CRAN (the Comprehensive R Archive Network) at

```
http://cran.r-project.org
```

Versions for Windows and Macintosh are both available. The default installation only comes with a few "packages". To get additional packages, click "Packages" on the top of window bar and then click "Install Packages". You can also install packages by typing the following in the R console:

```
install.packages("name of the package").
```

To use the packages, click "Load packages" under "Packages" tab.

The following tutorials are meant to get you started. More comprehensive introductions are readily available with help of Google.

1 Lab 1. Graphics and Checking Residuals

We have to have data to begin with. For the purpose of illustration, rather than real data sets, idealized data, which are produced by pseudo random generators, are used in this tutorial.

First let's try to generate 100 numbers according to standard normal (Gaussian) distribution.

```
> ?rnorm or help(rnorm)  # help on the use of function 'rnorm'
> x = rnorm(100)  # generate  100 random numbers from normal distribution
> qqnorm(x, main="Q-Q plot of data x")   #  Q-Q plot against normality
> help.search("")  #give you a list of functions and their associated packages
> help.search("distribution")   #functions related to "distributions"
```

The plot looks reasonably like a line, which confirms normality of the data x. The diagnostics of normality can also be done by plotting the density or histogram of the data.

```
> hist(x, nclass=10)
> plot(density(x))
```

Clearly the density estimator is a more refined tool than the histogram and is more informative than qqnorm. The density plot looks finer if we do:

```
> plot(density(x), xlab="x", ylab="density", type="l", main="density of data x")
```

You can also try xlab="" in the above and see what happens.

Now let us generate the data that have lighter tails than the normal distribution. We first generate 100 random numbers from uniform(-2,2) and repeat the above excise and see what happens.

```
> x1 = runif(100,-2, 2)
> qqnorm(x1)
> hist(x1, nclass=10)
> plot(density(x1, bw="sj"), type="l") # 'bw' refers to the smoothing bandwidth to be used.
```

The density estimation does not look close to uniform due to boundary effects and improper choice of bandwidth. The problem can easily be fixed, but this is beyond our scope of this class. Now, let us try on the heavy tailed Cauchy distribution.

```
> x2 = rcauchy(100)
> qqnorm(x2)
> hist(x2,nclass=10)
> plot(density(x2), type="l")
```

How do we putting the multiple plots in one picture? This can be done by changing some default parameters: try the following

```
> par(mfrow=c(3,2))  # putting 3*2=6 plots together
> qqnorm(x2)
> hist(x2, nclass=15)
> hist(x2, nclass=10)
> plot(density(x2))
......
```

To add some features to a plot:

```
> abline(0,1)   #add a line with intercept 0 and slope 1
> points(x2, dcauchy(x2))   # add a cauchy density estimated from x2 and evaluated at x2
> lines(x3,dcauchy(x3))
##‘lines’ is a generic function taking coordinates given in various ways and joining the
corresponding points with line segments.
```

You can also save the graphical output to a file by clicking the right-hand side of mouse.

2 Lab 2. Matrix operations

To input data directly from the console:

```
> A = scan()
1: 1 2 3 4 5 6 7 8 9   # press 1 to 9 and then ENTER
10:  # press ENTER to exist the scan environment.
Read 9 items
> A
[1] 1 2 3 4 5 6 7 8 9
```

There are other ways to call the ‘scan’ function. It is a flexible tool for importing data. We can use it to read in almost any type of data, numeric, character or complex, and it can be used for fixed or free formatted files.

```
> A = matrix(A, byrow=T, ncol=3)
## ‘ncol’ is the number of columns, and ‘byrow’ means that we exhaust an
## row in assigning values before going to the next row.
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> B = t(A)    # transpose of A
> B
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> A+10   # each element of A is added by 10
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   14   15   16
[3,]   17   18   19

> A + B       # matrix addition
> A %*% B     # matrix multiplication

> C = diag(1:3)  # create a diagonal matrix
> C
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3

> solve(C) %*% B   # C^{-1}B
```

Now let us try singular value decomposition:

```
> svd(A)   # decompose A = u %*% D %*% v', where D =diag(d)
$d
[1] 1.684810e+01 1.068370e+00 3.069525e-16

$u
           [,1]        [,2]       [,3]
[1,] -0.4796712  0.77669099  0.4082483
[2,] -0.5723678  0.07568647 -0.8164966
[3,] -0.6650644 -0.62531805  0.4082483
```

```
$v
           [,1]       [,2]       [,3]
[1,] -0.2148372 -0.8872307 -0.4082483
[2,] -0.5205874 -0.2496440  0.8164966
[3,] -0.8263375  0.3879428 -0.4082483


> S=svd(A)
> S$u %*% diag(S$d) %*% t(S$v)   ##this matrix is the same as A
```

3 Lab 3. Reading data and multiple regression

By far, we have been using computer generated data for analysis. This tutorial addresses the issue of reading data from other sources, and of performing regression analysis on these data.

First, download the Boston Housing data set from FIN504 course web. Suppose you have stored this data set at a local file.

```
> boston = scan("C:\\Users\\ABC\\Documents\\ORF504_2011S\\boston-housing.dat",skip=19)
# skip first 19 lines
> boston = matrix(boston, ncol=14, byrow=T)   #put the data in a matrix form
> boston[1:10,]  #display the first 10 rows
         [,1] [,2] [,3] [,4]  [,5]  [,6]  [,7]   [,8] [,9] [,10] [,11]  [,12] [,13] [,14]
 [1,] 0.00632 18.0 2.31    0 0.538 6.575  65.2 4.0900    1   296  15.3 396.90  4.98  24.0
 [2,] 0.02731  0.0 7.07    0 0.469 6.421  78.9 4.9671    2   242  17.8 396.90  9.14  21.6
 [3,] 0.02729  0.0 7.07    0 0.469 7.185  61.1 4.9671    2   242  17.8 392.83  4.03  34.7
 [4,] 0.03237  0.0 2.18    0 0.458 6.998  45.8 6.0622    3   222  18.7 394.63  2.94  33.4
 [5,] 0.06905  0.0 2.18    0 0.458 7.147  54.2 6.0622    3   222  18.7 396.90  5.33  36.2
 [6,] 0.02985  0.0 2.18    0 0.458 6.430  58.7 6.0622    3   222  18.7 394.12  5.21  28.7
 [7,] 0.08829 12.5 7.87    0 0.524 6.012  66.6 5.5605    5   311  15.2 395.60 12.43  22.9
 [8,] 0.14455 12.5 7.87    0 0.524 6.172  96.1 5.9505    5   311  15.2 396.90 19.15  27.1
 [9,] 0.21124 12.5 7.87    0 0.524 5.631 100.0 6.0821    5   311  15.2 386.63 29.93  16.5
[10,] 0.17004 12.5 7.87    0 0.524 6.004  85.9 6.5921    5   311  15.2 386.71 17.10  18.9

> dimnames(boston) = list(NULL, c("crim", "zn", "indus", "chas", "nox","rm", "age",
"dist", "rad", "tax", "ptratio", "b", "lstat", "medv"))
## name the column but not the rows.

> boston = data.frame(boston)
##A data frame is a kind of class in R, and it may for many purposes be regarded as a
##matrix with columns  possibly of differing  modes and attributes. It may be displayed
##in matrix form, and its rows and columns  extracted using matrix indexing conventions.
```

As an illustration, we regress the variable ‘medv’ against some others.

```
> attach(boston)
##The attach() function takes a ‘database’ such as a list or data frame as
##its argument.

> boston.lm = lm(medv ~ crim + indus + rm + age + dist + rad + tax + lstat)
> summary(boston.lm)

Call:
lm(formula = medv ~ crim + indus + rm + age + dist + rad + tax +
    lstat)

Residuals:
     Min       1Q   Median       3Q      Max
-16.6648  -3.2881  -0.8236   1.9821  28.3876

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.315263   3.605252   2.861  0.00440 **
crim        -0.100538   0.035974  -2.795  0.00539 **
indus       -0.106317   0.064680  -1.644  0.10086
rm           4.786106   0.443405  10.794  < 2e-16 ***
age         -0.019997   0.013966  -1.432  0.15281
dist        -1.044692   0.191049  -5.468 7.21e-08 ***
rad          0.143731   0.070532   2.038  0.04210 *
tax         -0.012390   0.004029  -3.075  0.00222 **
lstat       -0.576341   0.055260 -10.430  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 5.269 on 497 degrees of freedom
Multiple R-squared: 0.677,      Adjusted R-squared: 0.6718
F-statistic: 130.2 on 8 and 497 DF,  p-value: < 2.2e-16
```

4 Lab. 4: Matrix operation and multiple regression

This tutorial is to help us understand the object oriented language and its functionality. We will apply the matrix operations to do the multiple regression (you might want to consult an intermediate level mathematical statistics or linear regression textbook.) The following operations are done automatically by the function ‘lm’. But to get an insight for what this black box is about, we repeat the following exercise manually. Be sure to read the 2nd part of this exercise, as standardization of a data set is important in practice.

```
> beta = solve(t(X)%*%X)%*% t(X) %*% y #coef of LS fit
> resid = y - X %*% beta               # residual
> RSS = sum(resid^2)       #residual sum of squares
> sigma = sqrt(RSS/(length(y) - ncol(X))) # resid SD
> Cov = solve(t(X) %*% X)*sigma^2 # covariance matrix of beta
> Var = diag(Cov) # taking the diag elements
> SD = sqrt(Var)
> round(SD,4)
         crim  indus     rm    age   dist    rad    tax  lstat
3.6053 0.0360 0.0647 0.4434 0.0140 0.1910 0.0705 0.0040 0.0553

> tstat = beta / SD #t-statistics
> pvalue = 2*pt(-abs(tstat), 497)
> round(pvalue, 4) #p-value of t-statistics
        [,1]
      0.0044
crim  0.0054
indus 0.1009
rm    0.0000
age   0.1528
dist  0.0000
rad   0.0421
tax   0.0022
lstat 0.0000

Syy = (506-1) * var(y)
SSreg = Syy - RSS
> SSreg/Syy
[1] 0.6769684
> Fstat = SSreg/8/RSS*(506-9)
> 1-pf(Fstat,8,497)
[1] 0
```

Sometimes, we need to standardize each column so that it has mean zero and variance 1.

```
X = X[,-1] # delete the column 1
> dim(X)
[1] 506   8
> meanX = apply(X,2,mean) #mean of each column
> varX  = apply(X,2,var) #var of each column
> varX  = sqrt(varX)
> stdX  = (t(X) - meanX)/varX #meanX is a row matrix
> stdX = t(stdX)
> mean(stdX[,3])
[1] -1.056462e-16
> var(stdX[,3])
```

```
[1] 1


> cor(X)  #compute the correlation matrix
            crim      indus         rm        age       dist        rad        tax      lstat
crim   1.0000000  0.4065834 -0.2192467  0.3527343 -0.3796701  0.6255051  0.5827643  0.4556215
indus  0.4065834  1.0000000 -0.3916759  0.6447785 -0.7080270  0.5951293  0.7207602  0.6037997
rm    -0.2192467 -0.3916759  1.0000000 -0.2402649  0.2052462 -0.2098467 -0.2920478 -0.6138083
age    0.3527343  0.6447785 -0.2402649  1.0000000 -0.7478805  0.4560225  0.5064556  0.6023385
dist  -0.3796701 -0.7080270  0.2052462 -0.7478805  1.0000000 -0.4945879 -0.5344316 -0.4969958
rad    0.6255051  0.5951293 -0.2098467  0.4560225 -0.4945879  1.0000000  0.9102282  0.4886763
tax    0.5827643  0.7207602 -0.2920478  0.5064556 -0.5344316  0.9102282  1.0000000  0.5439934
lstat  0.4556215  0.6037997 -0.6138083  0.6023385 -0.4969958  0.4886763  0.5439934  1.0000000

> var(X)      #variance covariance matrix


> diag(var(X))  # variance of each column

> (varX)^2      #this number matches diag(var(X)) .
```

5 Lab 5. Regression Model Diagnostics and other operations

More knowledge on multiple regression analysis is required for this tutorial. Inexperienced readers might skip this part. Please read, process and attach the Boston housing data as in Lab 3.

```
 > X <- cbind(crim, indus, rm, dist, tax, ptratio,lstat)
> boston.ls <- lsfit(X,medv) #least-squares fit
> ls.print(boston.ls)
Residual Standard Error=5.0636
R-Square=0.7011
F-statistic (df=7, 498)=166.857
p-value=0

          Estimate Std.Err  t-value Pr(>|t|)
Intercept  24.6121  4.1341   5.9535   0.0000
crim       -0.0756  0.0333  -2.2683   0.0237
indus      -0.1299  0.0603  -2.1531   0.0318
rm          4.2260  0.4252   9.9377   0.0000
dist       -0.8827  0.1571  -5.6183   0.0000
tax        -0.0017  0.0023  -0.7521   0.4524
```

```
ptratio    -0.8344  0.1217  -6.8554   0.0000
lstat      -0.5909  0.0494 -11.9687   0.0000
```

The function ‘ls.diag’ computes basic statistics, including standard errors, t- and p-values for the regression coefficients. It takes as argument the result of ‘lsfit’.

```
boston.diag <- ls.diag(boston.ls)
leverage <- boston.diag$hat # ‘hat’ returns the diag entries of the hat matrix.

> sort(leverage)[-(1:500)] #show the highest 506-500 leverages
[1] 0.06679031 0.08442352 0.09280976 0.13958527 0.17046064 0.27067391


#Cook’s distance measures the effect of deleting a given observation.
cook <- boston.diag$cooks
> sort(cook)[-(1:500)] #show the highest 506-500 cook distances
[1] 0.06077564 0.06669306 0.07298053 0.08387908 0.11756544 0.23184746

> stdres <- boston.diag$std.res #standardized residuals
#internally studentized residuals

> studres <- boston.diag$stud.res #(externally) studendentized resituals
```

Now, let us check some diagnostic plots.

```
> par(mfrow=c(1,2))
> fitted <- medv - boston.ls$resid # fitted value of regression
>plot(fitted, studres) # fitted value vs. studentized resid
> identify(fitted, studres) # starting interactive graphical identification
#  of data: you can click anywhere on the plot and see what happens.

> plot(rm, studres) # rm vs studentized residuals
# recall that ‘rm’ is a feature.
```

We can use F-statistics to compare a model with its sub model:

```
> X1 <- cbind(crim,rm)
> boston1.ls <- lm(medv~X)
> boston2.ls <- lm(medv~X1)
> anova(boston1.ls,boston2.ls)
Analysis of Variance Table

Model 1: medv ~ X
Model 2: medv ~ X1
  Res.Df   RSS Df Sum of Sq      F    Pr(>F)
```

```
1    499 13974
2    503 19566 -4   -5592.1 49.923 < 2.2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

The extremely small p-value of the F-statistics indicates that we would prefer the full model over the simplified one.