

Splus is an object-oriented language. After double-click the splus icon, choose “windows” —> ”commands window” from the task bar. You can type help in Splus “Commands window” or from the task bar.

Lab 1. Graphic and Checking Residuals

Let us first try to generate 100 normal random and see what the qqnorm look likes:

```
> ?rnorm or help(rnorm) #help on the use of function rnorm
> x <- rnorm(100) #generate 100 random numbers from the normal dist
> qqnorm(x) #QQ-pllt against normality
> title("Q-Q plot of data x", "Figure 1") #adding title to the plot
```

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

```
> hist(x,nclass=10) #drawing histogram
> plot(density(x)) #plotting density of x
```

Clearly the density estimator is a more refined tool than the histogram and is more informative than qqnorm. The previous plot can look somewhat better by using some arguments on x .

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

You can try also `xlab = “ ”` in the above and see what happend.

```
> title("density of data x")
```

Now let us generate the data that have lighter tails than the normal distribution. We first generate 100 random number 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,width="sj"),type="l")
```

The density estimation does not look particularly well because of boundry effects. The problem can easily be remedied, but this is beyond our scope of this class.

Now, let us try on the Cauchy distribution.

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

Is it heavier tail than the normal distribution?

How do we putting the three pictures in one plot? This can be done by change some default parameters: try the following

```
> par(mfrow=c(3,2))    #3x2 plots
> qqnorm(x2)
> hist(x2,nclass=10)
> plot(density(x2),type="l")
```

To add some features to a plot, try the following:

```
> abline(0,0)          #adding a line with slope 0 and intercept 0
> points(x2,dcauchy(x2)) #add a cauchy density
> x3 <- seq(-6, 6, 0.1) #generating a sequence from -6 to 6 with spacing 0.1
> lines(x3, dcauchy(x3), lty=3) #add a cauchy density
```

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

Lab 2. Matrix operations

The create a vector on the screen, using the following command:

```
A <- scan()
1: 1 2 3 4 5 6 7 8 9 and type control-D
> A
[1] 1 2 3 4 5 6 7 8 9
> A <- matrix(A,byrow=T,ncol=3) #creating a matrix structure
#ncol=3 and hence nrow must be = 3
> 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
> A + 10          #each column is added by 10. How to added each row by 10?
      [,1] [,2] [,3]
[1,]   11   12   13
```

```

      [2,]  14  15  16
      [3,]  17  18  19
> A+B; A%*%B      #matrix addition and multiplication
> C <- diag(1:3)  #diagnoal 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 to do singular decomposition:

```

> svd(A)          # decompose A = v*D*u
$d:
[1] 1.684810e+01 1.068370e+00 1.963744e-16

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

$u:
      [,1]      [,2]      [,3]
[1,] -0.2148372  0.8872307  0.4082483
[2,] -0.5205874  0.2496440 -0.8164966
[3,] -0.8263375 -0.3879428  0.4082483
> C <- t(A) %*% A      # A' A
> svdC <- svd(C)
> u <- svdC$u          #take the component of u in svdC
> t(u)%*%u
      [,1]      [,2]      [,3]
[1,] 1.000000e+00 -2.220446e-16  1.665335e-16
[2,] -2.220446e-16  1.000000e+00 -5.551115e-17
[3,] 1.665335e-16 -5.551115e-17  1.000000e+00
> D <- diag(svdC$d)
> u %*% D %*% t(u)    ### This is the same as the matrix C

```

Lab 3. Reading data and multiple regression

Plus can read data from various format, including the .xls files. Click files —> import data —> from files. The Plus functions such as "scan" can also be used to read data into

Splus (useful for data that are irregularly formatted). Here, I give you an example (though the data are regularly formatted). If you want to try it, you need to download the data from my website (Boston Housing Data under 504).

```
> boston <- scan("t:\\sta3008\\boston-housing.dat", skip=19)
#skip the first 19 lines
> boston
> boston <- matrix(boston, ncol=14, byrow=T)           #put in matrix form
> boston[1:10,]                                       #check if data were arranged correctly.
> dimnames(boston) <- list(NULL, c("crim", "zn", "indus", "chas", "nox",
"rm", "age", "dist", "rad", "tax", "ptratio", "b", "lstat", "medv"))
#give each column a name
> boston <- data.frame(boston)
```

As an illustration, we fit the variable medv on a subset of variables:

```
> attach(boston)
> 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
-16.66	-3.288	-0.8236	1.982	28.39

Coefficients:

	Value	Std. Error	t value	Pr(> t)
(Intercept)	10.3153	3.6053	2.8612	0.0044
crim	-0.1005	0.0360	-2.7948	0.0054
indus	-0.1063	0.0647	-1.6437	0.1009
rm	4.7861	0.4434	10.7940	0.0000
age	-0.0200	0.0140	-1.4319	0.1528
dist	-1.0447	0.1910	-5.4682	0.0000
rad	0.1437	0.0705	2.0378	0.0421
tax	-0.0124	0.0040	-3.0751	0.0022
lstat	-0.5763	0.0553	-10.4296	0.0000

Residual standard error: 5.269 on 497 degrees of freedom

Multiple R-Squared: 0.677

F-statistic: 130.2 on 8 and 497 degrees of freedom, the p-value is 0

Lab 4: Matrix operation and multiple regression

To familiarize us understand the object oriented language and its functionality, let us apply the matrix operation to solve the multiple regression (you might need to consult a multiple regression book to understand the formulas). The following operations are done automatically by the function `lm`. But, to gain inside what the computer is doing, it is worthwhile to repeat the following exercise. Please do read the second part of this exercise.

```
> attach(boston)
> X <- cbind(1, crim, indus, rm, age, dist, rad, tax, lstat)
> y <- medv
> beta <- solve(t(X)%*%X)%*% t(X) %*% y      #coef of LS fit
> resid <- y - X %*% beta                  # residual
> RSS <- sum(resid^2)
> sigma <- sqrt(RSS/(length(y) - ncol(X))) # resid SD
> Cov <- solve(t(X) %*% X)*sigma^2        # covariance matrix of beta
> Var <- diag(Cov)                        # var of diag element
> SD <- sqrt(Var)
> round(SD,4)
[1] 3.6053 0.0360 0.0647 0.4434 0.0140 0.1910 0.0705 0.0040 0.0553
> tstat <- beta / SD                       #t-statistics
> round(tstat,4)
> pvalue <- 2*pt(-abs(tstat), 497)
> round(pvalue, 4)                         #p-value of t-statistics
> Syy <- 505 * var(y)
> SSreg <- Syy - RSS
> SSreg/Syy                                #multiple R^2
> 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. Let us practice this tricks.

```
> X <- X[,-1]                               # delete the column 1
502  0.06263 11.93 6.593 69.1 2.4786   1 273  9.67
503  0.04527 11.93 6.120 76.7 2.2875   1 273  9.08
504  0.06076 11.93 6.976 91.0 2.1675   1 273  5.64
505  0.10959 11.93 6.794 89.3 2.3889   1 273  6.48
506  0.04741 11.93 6.030 80.8 2.5050   1 273  7.88
> 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])
> var(stdX[,3])
> cor(X)                          #computing the correlation matrix
  crim 1.0000000  0.4065834 -0.2192467  0.3527343 -0.3796701  0.6255051
indus 0.4065834  1.0000000 -0.3916759  0.6447785 -0.7080270  0.5951293
  rm -0.2192467 -0.3916759  1.0000000 -0.2402649  0.2052462 -0.2098467
  age 0.3527343  0.6447785 -0.2402649  1.0000000 -0.7478805  0.4560225
> var(X)                          #variance covariance matrix
  crim 73.986578  23.992339 -1.3250378  85.405322  -6.8767215
indus 23.992339  47.064442 -1.8879566  124.513903 -10.2280975
  rm -1.325038  -1.887957  0.4936709  -4.751929  0.3036634
> diag(var(X))                    #variance of each column
[1] 7.398658e+01 4.706444e+01 4.936709e-01 7.923584e+02 4.434015e+00
[6] 7.581637e+01 2.840476e+04 5.099476e+01
> varX^2                          #variance of each column by using previous method
  crim  indus  rm  age  dist  rad  tax  lstat
73.98658 47.06444 0.4936709 792.3584 4.434015 75.81637 28404.76
50.99476

```

Lab 5. Regression Model Diagnostics and Other operations

This is deep into the multiple regression analysis. You may want to skip this part, particularly the second part of this hand out.

Let us revisit the boston housing data. There should be an object called "boston" in your Splus working directory. If **NOT**, try the following commands:

```

> boston <- scan("boston.housing.data", skip=19)      #skip the first 19
lines
> boston <- matrix(data, ncol=14, byrow=T)           #put in matrix form
> boston[1:10,]                                     #check if data were arranged
correctly.
> dimnames(boston) <- list(NULL, c("crim", "zn", "indus", "chas", "nox",
  "rm", "age", "dist", "rad", "tax", "ptratio", "b", "lstat", "medv"))
#give each column a name
> boston <- data.frame(boston)

```

Now, we try to analyze the data:

```

> attach(boston)

```

```

> boston[1:3,]                                # to see the name of variables
      crim zn indus chas   nox   rm age  dist rad tax ptratio   b lstat
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296   15.3 396.90 4.98
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242   17.8 396.90 9.14
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242   17.8 392.83 4.03

> X <- cbind(crim, indus, rm, dist, tax, lstat)
#take 6 variables as covariates
> boston.ls <- lsfit(X,medv)                    #least-squares fit
> ls.print(boston.ls)                          #printing out summary stat
Residual Standard Error = 5.0636, Multiple R-Square = 0.7011
N = 506, F-statistic = 166.857 on 7 and 498 df, p-value = 0

      coef std.err  t.stat p.value
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

> boston.diag <- ls.diag(boston.ls)           #getting raw materials for model diag
> ?ls.diag                                    #to see what the output are there?
> leverage <- boston.diag$hat                #take the output of leverages
> sort(leverage)                              #order the leverages
      123      365      121      122      369      368 124
0.04763923 0.04781834 0.04783256 0.04803416 0.05098243 0.05265302 0.05387807
      127      415      366      411      406      419      381
0.05428089 0.067434 0.08521597 0.09281169 0.1396106 0.1704624 0.2708908
#a few points have high leverages
> cook <- boston.diag$cooks                   #take Cook distance to see influences
> sort(cook)                                  #order the cook distances
      413      415      368      371      372      375      370
0.03773322 0.04090654 0.04635897 0.04707045 0.0519958 0.05319582 0.05324612
      365      381      373      366      369
0.05340162 0.06648093 0.07157419 0.09940017 0.2193488

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

```

```

> sort(abs(stdres))                                #sorting absolute value of standardized
#residuals
365      366      375      413      371      370      372      373
2.916654 2.921712 3.393918 3.434003 3.874674 4.25783 5.334387 5.534269
      369
5.715306

```

```

> studres <- boston.diag$stud.res                #(externally) studentized residuals
> sort(abs(studres))                            #similar to the above
      215      365      366      375      413      371      370      372
2.869968 2.938934 2.944119 3.430413 3.471906 3.930481 4.333156 5.48813
      373      369
5.706991 5.906561

```

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
> plot(rm, studres)                            # rm vs studentized residuals

```

Now, let us try to transform see if the rm needs a power transform:

```

> newvar <- rm*log(rm)                          # Arkinson's method
> X1 <- cbind(X, newvar)                        # adding the new variable rm*log(rm)
> boston.ls1 <- lsfit(X1,medv)                  # Least-squares fit
> ls.print(boston.ls1)
Residual Standard Error = 4.4126, Multiple R-Square = 0.7735
N = 506, F-statistic = 212.1014 on 8 and 497 df, p-value = 0

```

	coef	std.err	t.stat	p.value
Intercept	231.2306	16.7887	13.7730	0.0000
crim	-0.1047	0.0292	-3.5901	0.0004
indus	-0.0578	0.0529	-1.0930	0.2749
rm	-91.5372	7.6090	-12.0302	0.0000
dist	-0.5666	0.1392	-4.0707	0.0001
tax	-0.0033	0.0020	-1.6890	0.0918
ptratio	-0.5987	0.1077	-5.5587	0.0000
lstat	-0.6005	0.0430	-13.9544	0.0000
newvar	33.6263	2.6686	12.6005	0.0000

```
#### since the variable is highly significant, a transform is needed.
#### alpha = 4.2260/33.6263 + 1
```

Next, we illustrate the power transform of the response variables.

```
> Lambda <- seq(-2,2,by=0.5)           # possible values of lambda
> L      <- Lambda                     # creating a vector
> for(lambda in Lambda[-5])           # for lambda in -2.0 -1.5 -1.0
  #-0.5  0.5  1.0  1.5  2.0
  {ylambda <- medv^lambda              #power transform
  resid  <- lsfit(X, ylambda)$resid    #residuals of least-squares fit
  L[2*(lambda+2) + 1] <- sum(resid^2)  #RSS_lambda
  }

> L <- 506*log(abs(Lambda)) - 506/2*log(L) + (Lambda-1)*sum(log(medv))
                                     #compute the likelihood scores
> L
[1] -2849.539 -2638.204 -2465.497 -2348.491      NA -2321.197 -2392.053
[8] -2496.023 -2621.248

#####now deal the case with lambda = 0 #####
> residual <- lsfit(X,log(rm))$resid  # residuals of LS fit for transformed y
> L[5] <- -506/2*log(sum(residual^2)) - sum(log(medv))
                                     # likelihood score for lambda = 0

> L
[1] -2849.5394 -2638.2037 -2465.4971 -2348.4906 -885.1193 -2321.1972 -2392.0532
[8] -2496.0228 -2621.2476
> plot(Lambda, L)
```

To do variable selection, or compute the Cp score, try the following program. Note that the program can not handle more than 31 variables

```
> leaps(X,medv,nbest=1)                #variable selection
```