**Homework 4**

**BIST/STAT 6494 Fall 2017**

**Haim Bar**

**Due Date: October 8, 2017**

**(by 23:59, via HuskyCT)**

1. Run the program provided here to create a temporary SAS data set called School:

**data** school;

input Age Quiz : $1. Midterm Final;

/\* Add your statements here \*/

datalines;

12 A 92 95

12 B 88 88

13 C 78 75

13 A 92 93

12 F 55 50

13 B 88 82

;

**run**;

1. Use IF and ELSE IF statements, compute a new variable as follow: Grade (numeric), with a value of 6 if Age is 12 and a value of 8 if Age is 13.

if age eq **12** then grade = **6**;

else if age eq **13** then grade =**8**;

1. The quiz grades have numerical equivalents as follows: A = 95, B = 85, C = 75, D = 70 and F = 65. Using this information, compute a course grade (Course) as a weighted average of the Quiz (20%), Midterm (30%) and Final (50%). (Use IF and ELSE IF statements).

if quiz eq 'A' then quiz2 = **95**;

else if quiz eq 'B' then quiz2 = **85**;

else if quiz eq 'C' then quiz2 = **75**;

else if quiz eq 'D' then quiz2 = **70**;

else if quiz eq 'F' then quiz2 = **65**;

Course =**.2**\*quiz2+**.3**\*midterm+**.5**\*final;

1. Use SELECT statements to compute another new variable called FinalGRADE:

[97, 100] A+

[93, 97) A

[90, 93) A-

[87, 90) B+

[83, 87) B

[80, 83) B-

[77, 80) C+

[73, 77) C

[70, 73) C-

[67, 70) D+

[63, 67) D

[60, 63) D-

< 60 F

format FinalGRADE $2.;

select;

when (missing(course)) FinalGRADE = '.';

when (course lt **60**) FinalGRADE = 'F';

when (course lt **63**) FinalGRADE = 'D-';

when (course lt **67**) FinalGRADE = 'D';

when (course lt **70**) FinalGRADE = 'D+';

when (course lt **73**) FinalGRADE = 'C-';

when (course lt **77**) FinalGRADE = 'C';

when (course lt **80**) FinalGRADE = 'C+';

when (course lt **83**) FinalGRADE = 'B-';

when (course lt **87**) FinalGRADE = 'B';

when (course lt **90**) FinalGRADE = 'B+';

when (course lt **93**) FinalGRADE = 'A-';

when (course lt **97**) FinalGRADE = 'A';

when (course le **100**) FinalGRADE = 'A+';

otherwise;

end;

1. Use the IN operator to compute a new variable called STATUS: If FinalGRADE is F, STATUS is FAILED, otherwise it is PASSED.

if finalgrade in ('F') then STATUS ='FAILED';

else status='PASSED';

1. Use WHERE to create a subset of failed students.

**data** failed;

set school;

where status eq 'FAILED';

**run**;

1. Use the Sales data set:

*Tips: To copy the file path in Windows system, press shift and hold, right click the file, select “Copy as path”.*

1. List all the observations where Region is North and Quantity is less than 60.

**data** sales;

infile "P:\STAT-6494-Data Management in SAS and R\data\Sales.csv" dsd truncover firstobs=**2**;

input EmpID :**4.**

Name : $14.

Region : $5

Customer : $17.

Item : $5.

Quantity : **3.**

UnitCost : **5.**

TotalSales : **5.**;

**run**;

**proc** **print** data=sales;

where Region eq 'North' and quantity lt **60** and quantity is not missing;

**run**;

1. Include in this list any observations where the customer name (Customer) is Pet’s are Us or ID is 0177.

**proc** **print** data=sales;

where Region eq 'North' and quantity lt **60** and quantity is not missing

or customer eq "Pet's are Us" or empid eq **0177**;

**run**;

1. List all the observations where Name is Jason Nuygen and Region is East, or Name is not George Smith and Region is West.

**proc** **print** data=sales;

where name eq 'Jason Nuygen' and region eq 'East'

or name ne 'George Smith' and region eq 'West';

**run**;

1. List all the observations where there is missing data in Quantity, or Name contains Cost.

**proc** **print** data=sales;

where quantity is missing or name contains 'Cost';

**run**;

1. Input the dataset in question 1 into R.
2. Use if statements to compute a new variable as follow: Grade (numeric), with a value of 6 if Age is 12 and a value of 8 if Age is 13.

schooltxt <-"

12 A 92 95

12 B 88 88

13 C 78 75

13 A 92 93

12 F 55 50

13 B 88 82

"

school<-read.table(text=schooltxt, stringsAsFactors = F)

colnames(school)<-c("Age","Quiz","Midterm","Final")

n=length(school$Age)

school$Grade<-rep(0,n)

for (i in 1:n){

if (school$Age[i]==12) {school$Grade[i]=6}

if (school$Age[i]==13) {school$Grade[i]=8}

}

1. The quiz grades have numerical equivalents as follows: A = 95, B = 85, C = 75, D = 70 and F = 65. Using this information to compute a course grade (Course) as a weighted average of the Quiz (20%), Midterm (30%) and Final (50%).

school$Quiz2[school$Quiz=="A"]<-95

school$Quiz2[school$Quiz=="B"]<-85

school$Quiz2[school$Quiz=="C"]<-75

school$Quiz2[school$Quiz=="D"]<-70

school$Quiz2[school$Quiz=="F"]<-65

attach(school)

school$Course<- .2\*Quiz2+.3\*Midterm+.5\*Final

1. Compute another new variable called FinalGRADE:

[97, 100] A+

[93, 97) A

[90, 93) A-

[87, 90) B+

[83, 87) B

[80, 83) B-

[77, 80) C+

[73, 77) C

[70, 73) C-

[67, 70) D+

[63, 67) D

[60, 63) D-

< 60 F

school$FinalGRADE[school$Course<=100]<-'A+'

school$FinalGRADE[school$Course<97]<-'A'

school$FinalGRADE[school$Course<93]<-'A-'

school$FinalGRADE[school$Course<90]<-'B+'

school$FinalGRADE[school$Course<87]<-'B'

school$FinalGRADE[school$Course<83]<-'B-'

school$FinalGRADE[school$Course<80]<-'C+'

school$FinalGRADE[school$Course<77]<-'C'

school$FinalGRADE[school$Course<73]<-'C-'

school$FinalGRADE[school$Course<70]<-'D+'

school$FinalGRADE[school$Course<67]<-'D'

school$FinalGRADE[school$Course<63]<-'D-'

school$FinalGRADE[school$Course<60]<-'F'

school$FinalGRADE[is.na(school$Course)]<-'NA'

1. Check if there are any missing values in the dataframe.

is.na(school)

1. Use %in% create a subset of passed students.

school$status<-"PASSED"

school$status[school$FinalGRAD %in% c("F","NA")]<-"FAILED"

passed<-subset(school,school$status %in% c("PASSED"))

1. Repeat question 2 in R. (hint: If you want to test if characters is in a string, use grepl(). <https://stat.ethz.ch/R-manual/R-devel/library/base/html/grep.html>)

#4a

sales<-read.csv("P:/STAT-6494-Data Management in SAS and R/data/Sales.csv",header=T, stringsAsFactors = F )

attach(sales)

subset(sales,Region=="North" & Quantity < 60 & !is.na(Quantity))

#4b

subset(sales,Region=="North" & Quantity < 60 & !is.na(Quantity)

| Customer=="Pet's are Us" | EmpID==0177)

#4c

subset(sales, Name=="Jason Nuygen" & Region=="East"

| Name != "George Smith" & Region=="West")

#4d

subset(sales,is.na(Quantity) | grepl("Cost",Name))

1. **SAS:** Create and print a data set with variables N and LogN, where LogN is the natural logarithm of N.
2. Use a DO loop to create a table showing values of N and LogN for values of N going from 1 to 20.

**data** q5;

do N=**1** to **20**;

LogN=log(n);

output;

end;

**run**;

**proc** **print** data=q5;**run**;

1. Use a DO loop to create a table showing values of N and LogN for values of N going from 5 to 100 by 5.

**data** q5;

do N=**5** to **100** by **5**;

LogN=log(n);

output;

end;

**run**;

**proc** **print** data=q5;**run**;

1. **R:** Create and print a data set with variables N and LogN, where LogN is the natural logarithm of N.
2. Use a for loop to create a table showing values of N and LogN for values of N going from 1 to 20.

N<-c(1:20)

LogN<-rep(0,20)

for (i in 1:20){

LogN[i]<-log(N[i])

}

q6<-data.frame(N,LogN)

q6

1. Use a for loop to create a table showing values of N and LogN for values of N going from 5 to 100 by 5.

N<-seq(5,100,5)

LogN<-rep(0,length(N))

for (i in 1:length(N)){

LogN[i]<-log(N[i])

}

q6<-data.frame(N,LogN)

q6

1. **SAS:** Use an iterative DO loop to plot the following equation:

Y = 3\*x^2 – 5\*x + 10

Use values of x from 0 to 10, with an increment of .10. Use PROC PLOT to display the function.

**data** q7;

do x=**0** to **10** by **.10**;

Y=**3**\*x\*\***2**-**5**\*x+**10**;

output;

end;

**run**;

**proc** **plot** data=q7 ;

plot y\*x;

**run**;

1. **SAS:** Use an iterative DO loop to plot the following function:

Logit(p) = log(p / (1 – p))

Use values of p from 0 to 1 (with a point at every .05). Use PROC PLOT to plot your points.

**data** q8;

do p=**0** to **1** by **.05**;

Logitp=log(p/(**1**-p));

output;

end;

**run**;

**proc** **plot** data=q8;

plot Logitp\*p;

**run**;

1. **R**: Plot the function logit(p) using two ways – a for loop, and vectorized code.

#loop method

p<-seq(0,1,0.05)

logitp<-rep(NA,length(p))

for (i in 1:length(p)){

logitp[i]<-log(p[i]/(1-p[i]))

}

plot(p,logitp,type='l')

#vectorized code

p<-seq(0,1,0.05)

logitp<-log(p/(1-p))

plot(p,logitp,type='l')

1. **SAS:** You have the following seven values for temperatures for each day of the week, starting with Monday: 70, 72, 74, 76, 77, 78, and 85. Create a temporary SAS data set (Temperatures) with a variable (Day) equal to Mon, Tue, Wed, Thu, Fri, Sat, and Sun and a variable called Temp equal to the listed temperature values. Use a DO loop to create the Day variable.

**data** Temperatures;

do Day='Mon','Tue','Wed','Thu','Fri','Sat','Sun';

input Temp @;

output;

end;

datalines;

70 72 74 76 77 78 85

;

**run**;

1. **SAS:** You invest $1,000 a year at 4.25% interest, compounded quarterly. How many years will it take to reach $30,000? Do not use compound interest formulas. Rather, use “brute force” methods with DO WHILE and DO UNTIL statements to solve this problem. (in both ways)

\*with DO WHILE;

**data** interest;

r=**.0425**;

do while (x lt **30000**);

x+**1000**;

year+**1**;

i=**0**;

do while (i lt **4** and x lt **30000**);

i+**1**;

x=x\*(**1**+r/**4**);

output;

end;

end;

output;

**run**;

**proc** **print** data=interest;**run**;

\*with DO UNTIL;

**data** interest;

r=**.0425**;

do until (x ge **30000**);

x+**1000**;

year+**1**;

i=**0**;

do until (i ge **4** or x ge **30000**);

i+**1**;

x=x\*(**1**+r/**4**);

output;

end;

end;

output;

**run**;

**proc** **print** data=interest;**run**;

It takes 20 years.

1. **R:** Repeat the previous question in R. Use 2 different ways.

#1st way, use while

r=.0425

x=0

year=0

while (x<30000) {

x<-x+1000

year<-year+1

i=0

while (i<4 & x<30000){

i<-i+1

x<-x\*(1+r/4)

}

}

year

#2nd way, use repeat

r=.0425

x=0

year=0

repeat {

if (x>=30000) {break}

x<-x+1000

year<-year+1

i=0

repeat{

i<-i+1

x<-x\*(1+r/4)

if (i>=4 | x>=30000) {break}

}

}

year

1. **SAS:** Generate a table of integers and squares starting at 1 and ending when the square value is greater than 100. Use 2 different methods.

**data** q13;

do while (square le **100**);

integer +**1**;

square=integer\*\***2**;

output;

end;

**run**;

**proc** **print** data=q13;**run**;

\*2nd method, use do until;

**data** q13;

do until (square gt **100**);

integer +**1**;

square=integer\*\***2**;

output;

end;

**run**;

**proc** **print** data=q13;**run**;

1. **R:** Repeat the previous question in R. Use 2 different ways.

integer<-0

square<-0

while (square<=100){

integer<-integer+1

square<-integer^2

cat(integer,square,"\n")

}

#2nd way, use repeat

integer<-0

square<-0

repeat {

if (square>100) {break}

integer<-integer+1

square<-integer^2

cat(integer,square,"\n")

}

1. **R:** Evaluate the Fibonacci sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, …

In class, we used a for loop to solve this problem. But in practice it is not efficient. Please check this website <https://www.nayuki.io/page/fast-fibonacci-algorithms> for some fast algorithms of Fibonacci.

1. Use matrix exponentiation to evaluate . Make sure that your program is faster than the codes in notes (use system.time to check it). (Hint: eigenvalue decomposition).

n <- 1000

A<-matrix(c(1,1,1,0),nrow=2,ncol=2)

V<-eigen(A)$vectors

E<-diag(eigen(A)$values)

B<-V%\*%(E^(n-1))%\*%solve(V)

fib<-B[1,1]

fib

#check whether matrix exponentiation is faster

matrixexp<-function(n){

A<-matrix(c(1,1,1,0),nrow=2,ncol=2)

V<-eigen(A)$vectors

E<-diag(eigen(A)$values)

B<-V%\*%(E^(n-1))%\*%solve(V)

fib<-B[1,1]

return(fib)

}

forloop<-function(n){

fib <- rep(0,n)

fib[1] <- 1

fib[2] <- 1

for (i in 3:n) {

fib[i] <- fib[i-1] + fib[i-2]

}

return(fib[n])

}

n <- 1000

system.time(forloop(n))

system.time(matrixexp(n))

> system.time(forloop(n))

user system elapsed

0.02 0.00 0.01

> system.time(matrixexp(n))

user system elapsed

0 0 0

The note method use a longer time than my matrix exponentiation method.

1. Use fast doubling to evaluate . Make sure that your program is faster than the codes in notes. (Hint: try to avoid using recursive functions, use a loop instead)

fastdoub<-function(n){

fib<-rep(NA,n)

fib[1]<-1

fib[2]<-1

k<-floor(log(n,2))

p<-rep(NA,2\*k+1)

p[2\*k+1]<-n

for(i in k:1){

p[2\*i-1]<-floor(p[2\*i+1]/2)

p[2\*i]<-p[2\*i-1]+1

}

for(i in p[1:(2\*k)]){

fib[2\*i]<-fib[i]\*(2\*fib[i+1]-fib[i])

fib[2\*i+1]<-fib[i+1]^2+fib[i]^2

}

return(fib[n])

}

n <- 1000

system.time(fastdoub(n))

system.time(forloop(n))

> system.time(fastdoub(n))

user system elapsed

0 0 0

> system.time(forloop(n))

user system elapsed

0.02 0.00 0.01

The note method use a longer time than my program.

1. Repeat this procedure 1000 times, use system.time to find out the running time of each iteration and compare with the algorithm in notes. Then use a boxplot to illustrate your findings.

n <- 1000

m<-1000

t1<-rep(NA,m)

t2<-rep(NA,m)

t3<-rep(NA,m)

for (i in 1:m){

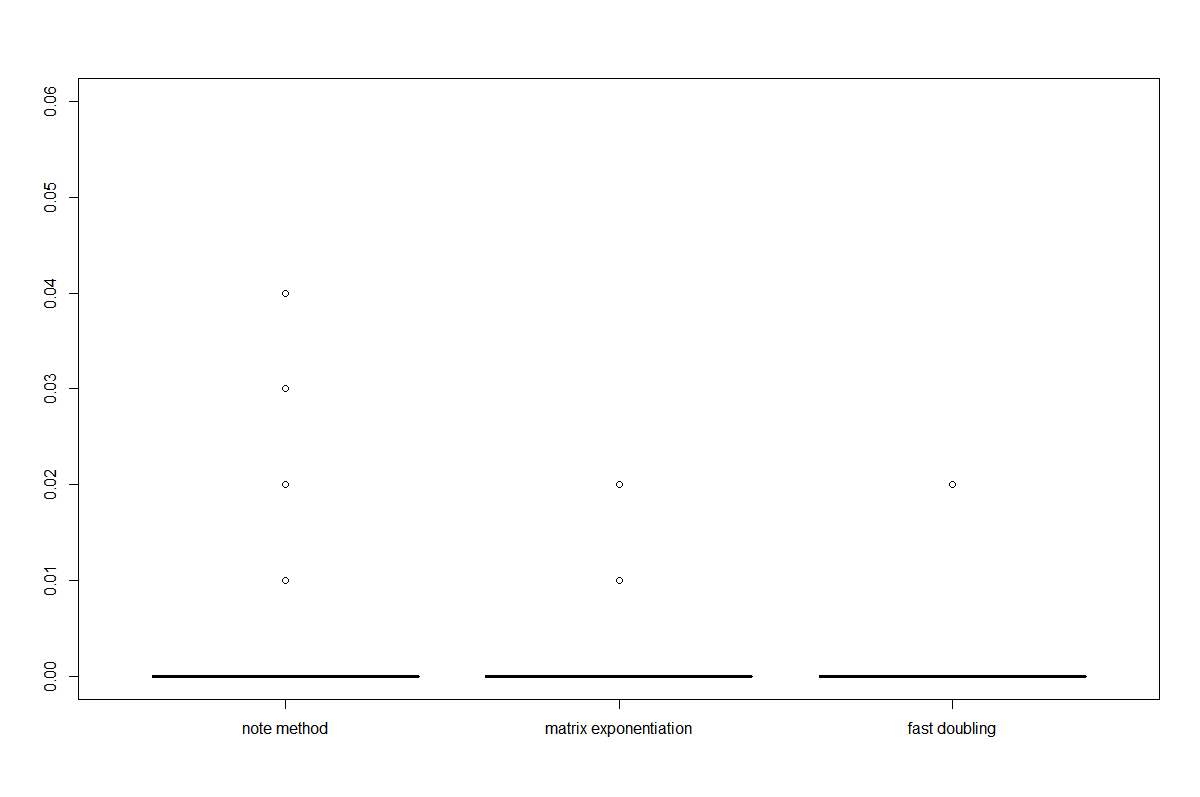
t1[i]<-system.time(forloop(n))[3]

t2[i]<-system.time(matrixexp(n))[3]

t3[i]<-system.time(fastdoub(n))[3]

}

boxplot(t1,t2,t3,names=c("note method","matrix exponentiation","fast doubling"), ylim=c(0,0.06))



The boxplot shows fast doubling is the fastest, and matrix exponentiation is faster note method. However, the difference is little.