Ch1

Sequence

```
> 1:20-rep(seq(0,9, by=3), rep(5,4))
 [1]  1  2  3  4  5  3  4  5  6  7  5  6  7  8  9  7  8  9 10 11
```

t-statistics

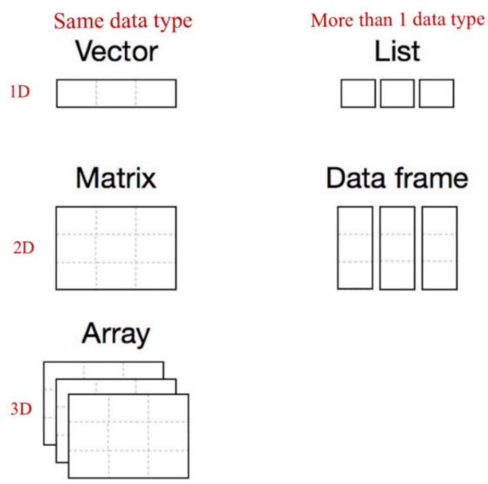
```
(t <- ((mean(x)-mean(y)))/(PooledSD*sqrt(1/n1+1/n2))) # t-statistic
qt(.975,n1+n2-2)  # critical value
(abs(t) > qt(.975,n1+n2-2)) # if TRUE, we should reject H_0
```

Object & Class & Data Type

object	possible modes	several modes possible in the same object ?
vector	numeric, character, complex, or logical	No
factor	numeric, or character	No
array	numeric, character, complex, or logical	No
matrix	numeric, character, complex, or logical	No
data.frame	numeric, character, complex, or logical	Yes
ts	numeric, character, complex, or logical	Yes
list	numeric, character, complex, logical, function, expression, or formula	Yes

`mode()` describe the **data type** used for storage, e.g., numeric, logical, character, etc.

`class()` describe the **object class** of the input variable, e.g., numeric, integer, list, matrix, factor, etc.



Ch2

Column Mean Using by()

```
by(data[,c(2,3)],data$Region,colSums)
```

Confidence Interval

```
n <- 1000
X <- rnorm(n)
Est <- mean(abs(X))     # estimate
SE <- sd(abs(X))/sqrt(n)  # standard error
CI95 <- c(Est-qnorm(0.975)*SE, Est+qnorm(0.975)*SE)
# 95% confidence interval
c(Est, sqrt(2/pi), CI95)
```

Aggregate()

To split the variable `year86` by `Region`, we can use

```
aggregate(year86~Region,d,mean)
```

```
aggregate(cbind(year86,year90)~Region+
dense,d,mean)
```

Ch3

Single Bar Chart

```
barplot(USPE[1:3,1],ylim=c(0,25),cex.names=0.8)
```

More about Bar Chart Arguments

```
a<-barplot(
  USPE[1:3,1:2], col=rainbow(3), ylim=c(0,50),
  beside=T, legend=T,
  args.legend=list(x="topright",bty="n",inset=c(-0.08, -0.02),cex=0.8),
  # Inset = Distance from Margin
  xlab="Year",
  ylab="Personal Expenditure",
  main="US Personal Expenditure in 1940 and 1945"
)
```

QQ-Plot for Uniform Distribution

```
n<-length(r)
r2<-sort(r)
i<-((1:n)-0.5)/n
q<-qunif(i)
plot(q,r2,main="Uniform QQ Plot")
abline(lsfit(q,r2), col="red")
```

Conditional Selection on Dataframe

`d[d$year86<d$year90,]` would select the observation according to this logical vector.

type = “” Argument for Plot

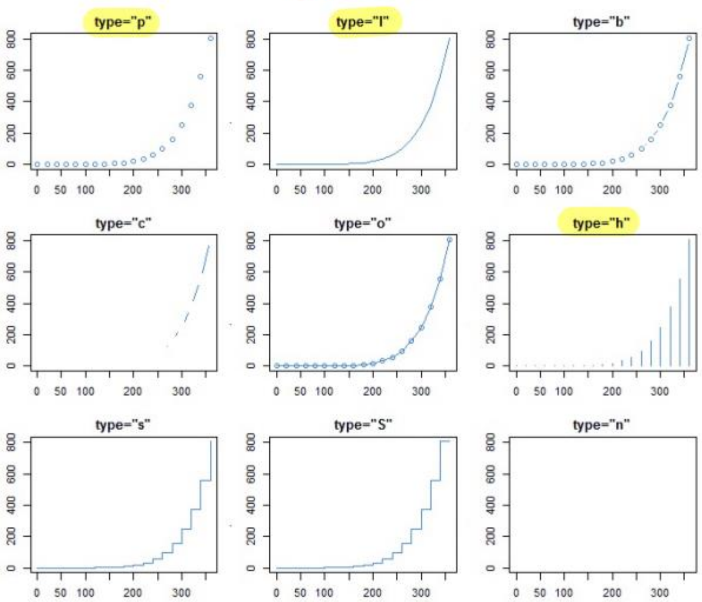
- “p”: is used for points plot
- “l”: is used for lines plot
- “b”: is used for both point plot and lines plot in a single place
- “c”: is used to join empty point by the lines
- “o”: is used for both lines and over-plotted point
- “h”: is used for ‘histogram plot’
- “s”: is used for stair steps
- “n”: is used for no plotting

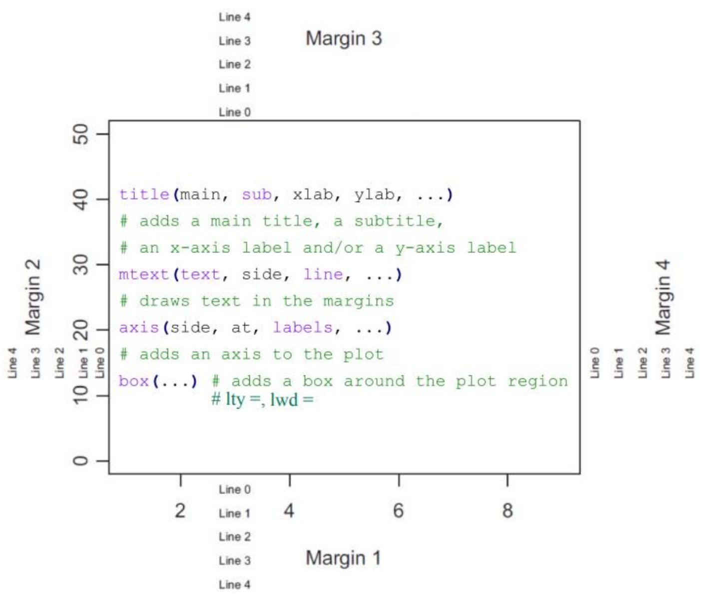
Time Series

One commonly used transformation in financial time series is $u_t = \ln(S_t/S_{t-1})$, where S_t is the stock price or index at time t.

```
u1<-log(lag(t1)/t1)
u2<-log(lag(t2)/t2)
u3<-log(lag(t3)/t3)
```

plot(type=)





Empty Plot

```
plot(0, 0, type="n", xlim=c(0,10), ylim=c(0,10),
        bty="n", xlab="", ylab="")
```

bty = o (default) / n / 7 / L / C / U

Ch4

Prime List

```
prime_list <- function(n) {
    if (n >= 2) {
        comp <- seq(2, n)
        primes <- c()
        for (i in seq(2, n)) {
            if (any(comp == i)) {
                primes <- c(primes, i)
                comp <- comp[(comp %% i) != 0]
            }
        }
        return(primes)
    } else {
        stop("Input value of n should be at least 2.")
    }
}
```

Fibonacci numbers

```
Fib1 <- 1
Fib2 <- 1
Fibonacci <- c(Fib1)
while (Fib2 < 300) {
    Fibonacci <- c(Fibonacci, Fib2)
    oldFib2 <- Fib2
    Fib2 <- Fib1 + Fib2
    Fib1 <- oldFib2
}
```

Compound Interest

```
r <- 0.11
period <- 1/12
debt_initial <- 1000
repayments <- 12
time <- 0
debt <- debt_initial
```

```
while (debt > 0) {
    time <- time + period
    debt <- debt*(1 + r*period) - repayments
}
cat('Loan will be repaid in', time, 'years\n')
```

Max Consecutive Appearance

```
max1<-function(v) {
    is_prev1<-FALSE         # initialize flag to False
    n1<-0; count<-0  # initialize counter
    for (i in v) {
        if ((i==1)&(is_prev1==TRUE)) {
            count<-count+1
            if (count>=n1) n1<-count
            next        # skip to next element in v
        }
        if ((i==1)&(is_prev1==FALSE)) {
            count<-1; is_prev1<-TRUE
            if (count>=n1) n1<-count
            next        # skip to next element in v
        }
        if ((i==0)&(is_prev1==TRUE)) {
            count<-0    # reset counter
            is_prev1<-FALSE
        }
    }
    return(n1)
}
```

Normal Table

```
y<-seq(0,3.4,0.1)
# define sequence of y from 0 to 3.4 with step 0.1
x<-seq(0,0.09,0.01)
# define sequence of x from 0 to 0.09 with step 0.01
z<-outer(y,x,"+")
# save the table to z, where z(i,j)=y(i)+x(j)
options(digits=4)       # specify output display to 4 decimal place
t<-pnorm(z)             # compute the left tail and save them to t
t<-rbind(x,t)           # add the first row to t
y<-c(0,y)               # add a zero to y
cbind(y,t)              # output the table
```

```
    y
x 0.0 0.0000 0.0100 0.0200 0.0300 0.0400 0.0500 0.0600 0.0700 0.0800 0.0900
  0.0 0.5000 0.5040 0.5080 0.5120 0.5160 0.5199 0.5239 0.5279 0.5319 0.5359
  0.1 0.5398 0.5438 0.5478 0.5517 0.5557 0.5596 0.5636 0.5675 0.5714 0.5753
  0.2 0.5793 0.5832 0.5871 0.5910 0.5948 0.5987 0.6026 0.6064 0.6103 0.6141
```

Ch5

Customise Operator

```r
> "%+-%" <- function(x,s) { c(x-s,x+s) }
> 3 %+-% 5
[1] -2  8
```

Formatting Output

```r
> sprintf("Pi is %f", pi)
# output real number with default option = 6 decimal places
[1] "Pi is 3.141593"
> sprintf("%.3f", pi)     # with 3 decimal places
[1] "3.142"
> sprintf("%5.1f", pi)    # fixed width=5 with 1 decimal places
[1] "  3.1"
> sprintf("%-10f", pi)    # left justified with fixed width=10
[1] "3.141593  "
> sprintf("%e", pi) # scientific notation
[1] "3.141593e+00"
```

Checking Symmetric Matrix

```r
n1<-dim(x)[1]     # get the row dimension of x
n2<-dim(x)[2]     # get the column dimension of x
if (n1!=n2) stop("Input matrix is not a square matrix")
for (i in 1:n1) {      # check if x is symmetric
      for (j in 1:i) {
            if (x[i,j]!=x[j,i])
                  stop("Input matrix is not a symmetric matrix")
      }
}
```

Sierpinski triangle

```r
set.seed(1234)     # set random seed
n<-5000            # number of points
for (i in 1:n) {
      col<-sample(c("b","g","r"),prob=c(1/3,1/3,1/3),size=1)
      # randomly pick a color
      if (col=="b") {           # color=blue
            x<-(x0+b1)/2        # mid-point between x0 and b
            y<-(y0+b2)/2
            points(x,y,pch=21,bg="blue")   # plot this point in blue
      }
      if (col=="g") {           # color=green
            x<-(x0+g1)/2        # compute mid-point bewtten x0 and g
            y<-(y0+g2)/2
            points(x,y,pch=21,bg="green")  # plot this point in green
      }
      if (col=="r") {           # color=red
            x<-(x0+r1)/2        # compute mid-point between x0 and r
            y<-(y0+r2)/2
            points(x,y,pch=21,bg="red")         # plot this point in red
      }
      x0<-x                     # update x0
      y0<-y                     # update y0
}
```

Recursive Function

```r
fac<-function(n){
      # factorial function, assume n is an integer > 0
      if (n<=2) return(n)
      else return(n*fac(n-1))
      # fac calls itself; fac(n)=n*fac(n-1)
}
```

Slash Matrix

```r
slash <- function(X) {
    m<-ncol(X)
    n<-nrow(X)
    (outer(1:n,1:m,"+")==min(m,n)+1)*X

    # for (i in 1:n) {
        # for (j in 1:m) {
            # if ((i+j)!=min(m,n)+1) X[i,j] <- 0
        # }
    # }
    # return(X)
}
```

Customise Sort()

```r
sort <- function(x) {
      # x is initially the input vector and will be
      # modified to form the output
      for(last in length(x):2) {
            for(first in 1:(last - 1)) {
                  if(x[first] > x[first + 1]) {
                        # swap the pair
                        save <- x[first]
                        x[first] <- x[first + 1]
                        x[first + 1] <- save
                  }
            }
      }
      return (x)
}
```

Additional Code for matrix generating

1. *Generating Matrix W/WO Loops (I)*

Code:

```
q1a <- function(n){
  M <- matrix(0,nrow=n,ncol=n)
  for (i in 1:n){
    for(j in 1:n){
      for(x in 2:n){
        if((i+j)==x |((i+j)==2*n-x+2)){
          M[i,j]<-x-1
        }
        if((i+j)==(n+1)){
          M[i,j]<-n
        }
      }
    }
  }
  return(M)
}
```

```
q1b <- function(n) {
  row_indices <- matrix(1:n, nrow = n, ncol = n, byrow = TRUE)
  col_indices <- matrix(1:n, nrow = n, ncol = n, byrow = FALSE)

  M1 <- row_indices + col_indices
  M2 <- n - abs(M1 - (n + 1))

  M <- pmin(M1, M2)

  return(M)
}
```

Output:

```
> q1a(6)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    2    3    4    5    6    5
[3,]    3    4    5    6    5    4
[4,]    4    5    6    5    4    3
[5,]    5    6    5    4    3    2
[6,]    6    5    4    3    2    1
```

2. *Generating Matrix W/WO Loops (II)*

Code:

```
q2a <- function(n){
  M <- matrix(0,ncol=n,nrow=n)
  for(i in 1:n){
    for(j in 1:n){
      M[i,j] <- i +j -1
    }
  }
  return(M)
}
```

```
q2b <- function(n) {
  row_indices <- matrix(1:n, nrow = n, ncol = n, byrow = TRUE)
  col_indices <- matrix(1:n, nrow = n, ncol = n, byrow = FALSE)
  M <- row_indices + col_indices - 1
  return(M)
}
```

Output:

```
> q2a(6)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    2    3    4    5    6    7
[3,]    3    4    5    6    7    8
[4,]    4    5    6    7    8    9
[5,]    5    6    7    8    9   10
[6,]    6    7    8    9   10   11
```

3. Reflecting

Code:

```
mirror <- function(A, m) {
  if (m == 1) {
    A <- A[, ncol(A):1] # Left/Right
  } else if (m == 2) {
    A <- A[nrow(A):1, ] # Up/Down
  } else {
    stop("m must be either 1 or 2")
  }
  return(A)
}
```

Output:

```
> mirror(q1a(6),1)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    6    5    4    3    2    1
[2,]    5    6    5    4    3    2
[3,]    4    5    6    5    4    3
[4,]    3    4    5    6    5    4
[5,]    2    3    4    5    6    5
[6,]    1    2    3    4    5    6
```

```
> mirror(q1a(6),2)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    6    5    4    3    2    1
[2,]    5    6    5    4    3    2
[3,]    4    5    6    5    4    3
[4,]    3    4    5    6    5    4
[5,]    2    3    4    5    6    5
[6,]    1    2    3    4    5    6
```

4. Lower & Upper Triangular Matrix

Code:

```
lower_matrix <- function(n) {
  M <- matrix(0, nrow = n, ncol = n)
  count <- 1
  for (i in 1:n) {
    for (j in 1:i) {
      M[i, j] <- count
      count <- count + 1
    }
  }
  return(M)
}
```

```
upper_matrix <- function(n) {
  M <- matrix(0, nrow = n, ncol = n)
  count <- 1
  for (i in 1:n) {
    for (j in i:n) {
      M[i, j] <- count
      count <- count + 1
    }
  }
  return(M)
}
```

Output:

```
> lower_matrix(4)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    2    3    0    0
[3,]    4    5    6    0
[4,]    7    8    9   10
```

```
> upper_matrix(4)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    0    5    6    7
[3,]    0    0    8    9
[4,]    0    0    0   10
```

5. Getting User Input

input <- readline(prompt=”Enter a number”)

input <- as.integer(input)

Ch6

Matrix Operation

%*% = Matrix Multiplication / Inner Product

%o% = Outer Product

Usage of diag(x)

1. If x is a vector, it will create a diagonal matrix with diagonal entries: $x_1, x_2, \ldots, x_n$
2. If x is a matrix, it will extract the diagonal entries as a vector.
3. If x is an integer, it will generate a x-by-x identity matrix.
4. Replacement form: diag(x) <- v

   Replace the diagonal elements of x by vector v.

Markov Chain:

```
T <- matrix(c(.5,.2,.3,.2,.6,.2,0,.1,.9), nrow=3, byrow=TRUE)

# Q1(b)
p <- c(1,0,0) # Since given X_1 = 1 # OR p <- diag(3)
T3 <- p
for (i in 1:3) {
  T3 <- T3 %*% T
}
T3[3] # X_3
```

```
# Q1(d)
eig1<-eigen(t(T))
(eig1$vectors[,1]/sum(eig1$vectors[,1]))
(eig1$vectors[,1]/sum(eig1$vectors[,1]))[2]
```

Generating upper triangular matrix in one line command:

```
> n<-4
> (u<-(1-outer(1:n, 1:n, "-"))*outer(1:n, 1:n, "<="))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    0    1    2    3
[3,]    0    0    1    2
[4,]    0    0    0    1
```

```
> u+t(u)-diag(diag(u))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    1    2    3
[3,]    3    2    1    2
[4,]    4    3    2    1
```

```
> abs(outer(1:n, 1:n, "-"))+1
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    1    2    3
[3,]    3    2    1    2
[4,]    4    3    2    1
```

Using solve() function:

x <- solve(A,b)   # Solve Linear Equation

Ax=b;   A: matrix; x: unknown vector;   b: constant vector

iden <- diag(c(1,1))

solve(A,iden) / solve(A)   # Find matrix inverse

Calculating Summation W/O for loops:

$$\sum_{i=1}^{25}\left(\frac{2^i}{i}+\frac{3^i}{i^2}\right)$$

```
> i<- 1:25
> sum((2^i)/i+3^i/(i^2))
```

$$\sum_{i=1}^{20}\sum_{j=1}^{5}\frac{i^4}{3+j}$$

```
> sum((1:20)^4)*sum(1/(4:8))
[1] 639215.3
> sum(outer((1:20)^4,4:8,"/"))
```

Finding roots of nonlinear equations:

uniroot():

fx <- function(x) {…}

interval <- c(0.0001,1) / c(-10,10)

uniroot(fx, interval)

```
$root              # answer
[1] 0.06762566
$f.root        # function value at root
[1] -0.005176046
$iter              # number of iteration
[1] 4
$estim.prec   # precision of solution
[1] 6.103516e-05
```

Finding roots of nonlinear equations:

Self-defined function:

```
bisection<-function(f, x1, x2, n = 1000, err = 1e-05) {
      f1 <-f(x1); f2 <-f(x2)
      if (f1==0) return(x1)
      else if (f2==0) return(x2)
      else if (f1*f2>0) stop("Roots may not exist in range")
      else {
            x <-(x1+x2)/2; fx <-f(x)
            i <-0
            while ((abs(fx)>err)&(i<=n)) {
                  if (fx*f2>0) {
                        x2 <-x
                  } else if (fx*f1>0) {
                        x1 <-x
                  }
                  x <-(x1+x2)/2; fx<-f(x)
                  i <-i+1
            }
      }
      return(x)
}
bisection(fx,0.0001,1,1000,10^-5)
```

Differentiation & Integration:

D(expr, “x”)   # f’(x)

D(D(expr, “x”))   # f”(x)

integrate(func, lb, ub)

```
fx <- expression(x^2 + sin(x))      # f(x)
dfdx <- D(fx, "x")                  # f'(x)
x<-5
eval(fx)                            # Compute f(5)
eval(dfdx)                          # Compute f'(5)
```

```
stdnorm<-function(x) { exp(-x^2/2)/sqrt(2*pi) }
# define standard normal density
integrate(stdnorm, -Inf, 0)
# integrate from -infinity to 0
0.5 with absolute error < 4.7e-05
```

Univariate optimization:

1 Variable: optimize(function, interval)

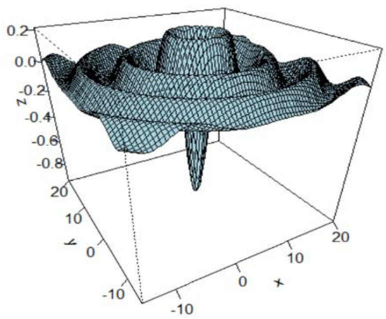
Using g(x) = f(x)*f(x) = 0 to find the minimum value (default).

Add parameter maximum = TRUE to find maximum value.

2 Variables: optim(initial value, function)

Create a 3D plot to illustrate the 2 variables objective function:

```
fxy<-function(x,y) {
# re-defnine fx using two arguments x and y
   p<-sqrt((x-5)^2+(y-5)^2)
   -sin(p)/p }
x<-seq(-15,20,by=0.5)
# define a sequence of x and y near the solution
y<-x
z<-outer(x,y,fxy)
# create a 71x71 matrix whose element is fxy(x,y)
persp(x,y,z,theta=-30, phi=30, col="lightblue",
ticktype="detailed")
```



phi: viewing angle

theta: horizontal rotation

ticktype: axis number

Ch9 SAS

Informat

Informat	Definition	Width range	Default width
Character			
$CHAR*w*.	Reads character data—does not trim leading or trailing blanks	1–32,767	8 or length of variable
$UPCASE*w*.	Converts character data to uppercase	1–32,767	8
$*w*.	Reads character data—trims leading blanks	1–32,767	none
Numeric			
COMMA*w.d*	Removes embedded commas and $, converts left parentheses to minus sign	1–32	1
COMMAX*w*.	Like COMMA*w.d* but reverses role of comma and period	1–32	1
PERCENT*w*.	Converts percentages to numbers	1–32	6
w.d	Reads standard numeric data	1–32	none

Informat	Input data	INPUT statement	Results
Character			
$CHAR*w*.	`my cat` `   my cat`	`INPUT Animal $CHAR10.;`	`my cat` `   my cat`
$UPCASE*w*.	`my cat`	`INPUT Name $UPCASE10.;`	`MY CAT`
$*w*.	`my cat` `   my cat`	`INPUT Animal $10.;`	`my cat` `my cat`
Numeric			
COMMA*w.d*	`$1,000,001` `(1,234)`	`INPUT Income COMMA10.;`	`1000001` `-1234`
COMMAX*w*.	`$1.000.001` `(1.234,25)`	`INPUT Value COMMAX10.;`	`1000001` `-1234.25`
PERCENT*w*.	`5%` `(20%)`	`INPUT Value PERCENT5.;`	`0.05` `-0.2`
w.d	`1234` `-12.3`	`INPUT Value 5.1;`	`123.4` `-12.3`

Date, Time, and Datetime[8]			
ANYDTDTE*w*.	Reads dates in various date forms	5–32	9
DATE*w*.	Reads dates in form: *ddmmmyy* or *ddmmmyyyy*	7–32	7
DATETIME*w*.	Reads datetime values in the form: *ddmmmyy hh:mm:ss.ss*	13–40	18
DDMMYY*w*.	Reads dates in form: *ddmmyy* or *ddmmyyyy*	6–32	6
JULIAN*w*.	Reads Julian dates in form: *yyddd* or *yyyyddd*	5–32	5
MMDDYY*w*.	Reads dates in form: *mmddyy* or *mmddyyyy*	6–32	6
STIMER*w*.	Reads time in form: *hh:mm:ss.ss* (or *mm:ss.ss*, or *ss.ss*)	1–32	10
TIME*w*.	Reads time in form: *hh:mm:ss.ss* (or *hh:mm*)	5–32	8

Date, Time, and Datetime			
ANYDTDTE*w*.	`1jan1961` `01/01/61`	`INPUT Day ANYDTDTE10.;`	`366` `366`
DATE*w*.	`1jan1961` `1 jan 61`	`INPUT Day DATE10.;`	`366` `366`
DATETIME*w*.	`1jan1960 10:30:15` `1jan1961,10:30:15`	`INPUT Dt DATETIME18.;`	`37815` `31660215`
DDMMYY*w*.	`01.01.61` `02/01/61`	`INPUT Day DDMMYY8.;`	`366` `367`
JULIAN*w*.	`61001` `1961001`	`INPUT Day JULIAN7.;`	`366` `366`
MMDDYY*w*.	`01-01-61` `01/01/61`	`INPUT Day MMDDYY8.;`	`366` `366`
STIMER*w*.	`10:30` `10:30:15`	`INPUT Time STIMER8.;`	`630` `37815`
TIME*w*.	`10:30` `10:30:15`	`INPUT Time TIME8.;`	`37800` `37815`

Ch9 SAS

Notations	*Meaning / Usage*
|	Choice of items
…	Item may be repeated
[…]	Optional items
{…}	Define a item

Input Notations	*Meaning / Usage*
$	Character variable
&	Indicating there are spaces in the variable
/	Jump to / Create next line
@@	≥ 1 observation in a single line / Put at the end of IS
CARDS4 ;	Input data with ‘ ; ’, need to end with ‘ ;;;; ’

Input Statements	*Meaning / Usage*
LENGTH *x* $ 10 ;	Defining a char var *x* with length 10 (default = 8)
INPUT name $ 1-5 ;	Column input format
INPUT name $10. ;	Character variable with length = 10
INPUT height 5.1 ;	Numeric variable with length = 5, decimal place = 1
INPUT Income COMMA10. ;	Numeric variable with length = 5, separated by comma
INPUT Day ANYDTDTE10. ;	Number of days after 1st Jan 1960
INPUT Day DATE10. ;	

Column / Mixed Input	*Meaning*
INPUT Name $16. Age 3. +1 Type $1.	Next 3 cols of Col 16 is Age, then skip 1 line
INPUT @17 Age 2.	2 cols from Col 17 is Age
INPUT salary : comma10.	Read up to 10 char width, or a blank space
DSD;	2 consecutive delimiters as a missing value
	Remove quotation marks
	Set default delimiters to a comma
DLM = ‘/’	Set delimiters to ‘/’

Ch10 Data Manipulation

	Symbol	Alt. symbol	Meaning
Arithmetic operators :	+		(prefix) makes value positive, e.g. +2
	-		(prefix) makes value negative, e.g. -2
	**		Exponentation, e.g. 2**4 (means 2^4)
	*		Multiplication, e.g. 2*4 gives 8
	/		Division, e.g. 4/2 gives 2
	+		Addition, e.g. 4+2 gives 6
	-		Subtraction, e.g. 4-2 gives 2
	<>		Maximum, e.g. 2<>4 gives 4
	><		Minimum, e.g. 2><4 gives 2
Character string operation:	||		Concatenation, e.g. 'ab'||'cde' gives 'abcde'

	Symbol	Alt. symbol	Meaning
Comparison Operators :	=	EQ	Equal, e.g. a = b , a eq b (gives value TRUE if and only if a = b)
	^=	NE	Not equal, e.g. a ^= b, a ne b (gives value TRUE if and only if a is not equal to b)
	>	GT	Greater than, e.g. a > b, a gt b
	>=	GE	Greater than or equal, e.g. a >= b, a ge b
	<	LT	Less than, e.g. a < b, a lt b
	<=	LE	Less than or equal, e.g. a <= b, a le b
Logical (Boolean) operators :	^	NOT	(prefix) negation, e.g. ^(a+b = 4)
	&	AND	And, e.g. a = 1 & b = 2 , a = 1 and b = 2
	|	OR	Or, e.g. a = 1 | b = 2, a = 1 or b = 2
Other operator :		IN	List membership, e.g. a in (6, 7, 8) (it gives TRUE value if and only if a is 6, 7 or 8), region in ('NE', 'W', 'S')

Operator	Associativity
Function terms (such as SIN, LOG and SQRT)	
** , + (prefix) , -(prefix), NOT, <>, ><	Right to left
* , /	Left to right
+ (infix), - (infix)	Left to right
||	Left to right
Comparisons (such as >, >=)	Left to right
AND	Left to right
OR	Left to right

Variable List

Shortcuts	*Meaning*
varm - varn	varm, var(m+1), … var(n-1), varn
vara -- varb	All variables physically between a & b
vara -numeric - varb	All numeric variables between a & b
vara -character - varb	All character variables between a & b
NUMERIC	All numeric variables in the dataset
CHARACTER	All character variables in the dataset
ALL	All variables in the dataset

Remarks: Use _ALL_ will generate 2 more variables: _ERROR_ & _N_ # Observations

Built-in Functions

1 Argument: funcname(arg)

>1 Arguments: funcname(arg, arg, …) / funcname(OF arg_list)

Eg. sum(x_1,x_2,x_3), sum(OF x_1-x_3), sum(OF _numeric_)

Arithmetic Functions:

sqrt(num)	min([OF] arg_list)	exp(power)
abs(num)	max([OF] arg_list)	log(arg)
sign(num)	sin/cos/tan()	floor(arg)
mod(num1, num2)	arsin/arcos/atan()	round(arg [,amount])

Statistical Functions:	*Meaning / Applications*
N([OF] arglist)	# Non-missing values
NMISS([OF] arglist)	# Missing values
SUM([OF] arglist)	Sum of arguments, ignore missing value
MEAN([OF] arglist)	Mean of arguments
STD([OF] arglist)	Standard Deviation
PROBBNML(p,n,x)	P(B(n,p)) <= x
PROBNORM(arg)	P (Z <= arg)
PROBIT(p)	P(Z < x) = p

Random Number Generation	*Meaning / Applications*
RANBIN(seed,n,p)	Generate a binomial random number
RANNOR(seed)	Generate a standard normal random number
RANUNI(seed)	Generate a U(0,1) random number

Date & Time Functions	*Outputs*
TODAY()	# Days after 1 Jan 1960 (SAS Date)
TIME()	# Seconds after 00:00
YEAR(arg)	arg: SAS Date, return calendar year
MONTH(arg)	arg: SAS Date, return calendar month
DAY(arg)	arg: SAS Date, return calendar day
MDY(m,d,y)	Convert calendar date to SAS date
WEEKDAY(arg)	Return numeric day of week (1: Sunday)

Extra Date & Time Functions:

Format	*Example*	*Output*
‘ddmmmyyyy’d	‘11jan1999’	Return SAS Date
’hh:mm’t	’10:30’t	# Seconds after 00:00
‘ddmmmyyyy:hh:mm’dt	‘2Jan1960:10:30’dt	# Seconds after 1 Jan 1960

Character Functions	*Meaning / Applications*
LENGTH(string)	Return position of rightmost non-blank character
LOWCASE(string)	Convert the string to lower case
UPCASE(string)	Convert the string to upper case
INDEX(source, arg)	Return position of the arg, 0 if not found
INDEXC(source, arg)	Return 1st position of any char in arg
INDEXW(souces, arg)	Return position of exact word as arg
SUBSTR(arg, start, [,length])	Extract substring
TRIM(arg)	Remove trailing blanks
SCAN(arg, n)	Return the n-th word, +n: from left, -n: from right

Default delimiters:

```
blank ! $ % & ( ) * + , - . / ; < ^ |
```

Control flow:

If Statement: Single Action

IF condition THEN action1; [ELSE action2;]

If Statement: Multiple Actions

IF condition

THEN DO; action1; action2; END;

ELSE DO; action3; action4; END;

Select (Switch):

1.

SELECT:

WHEN (condition1) action1;

WHEN (condition2) action2;

OTHERWISE [action3];

END;

2.

SELECT(A);

WHEN (1) action1;

WHEN (2) action2;

OTHERWISE [action3];

END;

Do-loops (For-loops):

1. ***Index takes numeric value only, default increment = 2***

DO index = begin TO end [BY increment];

< Operations >

END;

Skip observations / Partition of interval:
DO X = 0, 1 TO 4 BY 0.02;

2. ***Index takes numeric or character value***

DO index = value [{, value} …];

< Operations >

END;

Character variable:
DO x = ‘A’, ‘B’, ‘C’, ‘D’;

Output Statement:

Put it at the end of the DO-loop (operations) → Output every pair of value

Without it → Only output the last pair of values

Do-While loops (While-loops):

DO WHILE (condition);

< Operations >

END;

Ch11 Controlling Outputs

IF (condition1 AND condition 2) THEN OUTPUT; # No implicit output after OUTPUT

IF (condition) THEN DELETE; # Need OUTPUT afterwards.

WHERE Statement:

WHERE condition;

Implicit output for data satisfies the condition.

```
DATA SAMPLE;
SET STD(FIRSTOBS = 10 OBS = 17
WHERE=(MATH > 70));
RUN;
```

This operates first

Difference between IF and WHERE statements:

IF statement	***WHERE statement***
Read all observations then select	Select data satisfies condition, then read whole observation
Can select extra variables	Only select existing variables

Output 2 datasets:

```
DATA MALE FEMALE;
  SET SCHOOL.CLASS;
  IF GENDER = 'M' THEN OUTPUT MALE;
  ELSE IF GENDER = 'F' THEN OUTPUT FEMALE;
RUN;
```

STOP Statement:

IF (condition) THEN STOP;

Stop the execution when the condition is TRUE, the code afterwards will not be executed.

Dataset Options:

1. ***KEEP:***

KEEP = varlist;

DATA dsname (KEEP = A B);

SET dsname (KEEP = A B);

2. ***DROP:***

DROP = varlist;

DATA dsname (DROP = A B);

SET dsname (DROP = A B);

3. ***RENAME:***

SET ABC (RENAME = (old=new n=m)); # Contents of ABC won't change.

4. ***Using WHERE statement:***

DATA ABC (WHERE = (condition));

SET ABC (WHERE = (condition));

Variable dropped cannot be used for condition checking

5. ***Observation Number:***

SET ABC (FIRSTOBS=10 OBS=20);

Read observation from 10 – 20 / The option order does not matter

6. ***Label:***

DATA ABC (LABEL = "Fuck SAS");

7. ***Use in PROC***

The dataset options can be used in PROC.

PROC PRINT DATA = AA (OBS=10 WHERE(REGION=1));

RUN;

Print the first 10 observations satisfies the condition.

: modifier with informat

When data are not aligned in columns but we need additional instructions that only informats can provide, a : modifier would be useful.

A : modifier with an informat enables SAS to do the following:

- Treat the current field as a delimited field
- Apply an informat to the field, ignoring the width

```
DATA Employee1;
     INPUT name $ salary:comma10. state $;
     * list input;
CARDS;
Ted $2.345 Georgia
Sam $222,345 Florida
RUN;
```

```
DATA kids;
   INFILE "D:\SAS\kids.dat" DSD;
   * INFILE 'D:\SAS\kids.dat' DLM=','; /*Does not work*/
   INPUT name $
         siblings
         bdate : mmddyy10.
         allowance : comma2.
         hobby1 : $10.
         hobby2 : $10.
         hobby3 : $10.;
RUN;
```

```
Chloe,,11/10/1995,,Running,Music,Gymnastics
Travis,2,1/30/1998,$2,Baseball,Nintendo,Reading
Jennifer,0,8/21/1999,$0,Soccer,Painting,Dancing
```

```
DATA kids_a;
   INFILE 'D:\SAS\kids_a.dat' DLM='/' DSD;
   INPUT name $
         siblings
         bdate : mmddyy10.
         allowance : comma2.
         hobby1 : $10.
         hobby2 : $10.
         hobby3 : $10.;
RUN;
```

```
Chloe/2/"11/10/1995"/$5/Running/Music/Gymnastics
Travis/2/"1/30/1998"/$2/Baseball/Nintendo/Reading
Jennifer/0/"8/21/1999"//Soccer/Painting/Dancing
```

Extra Example on Data Manipulation

Example 19

It can be shown that

$$\frac{2}{\pi} = \underbrace{\sqrt{\frac{1}{2}}}_{A_1} \times \underbrace{\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}_{A_2} \times \underbrace{\sqrt{\frac{1}{2}+\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}}_{A_3} \times \cdots$$

Define

$$B_1 = \sqrt{\frac{1}{2}},\ \ B_2 = \sqrt{\frac{1}{2}} \times \sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}, \ldots$$

We have B_n converges to $2/\pi$ as n approaches infinity. Write a SAS program to compute $C = B_{20}$ and $D = 2/B_{20}$ (which should be close to π).

Let $A_0 = 0, B_0 = 1$

$$A_j = \sqrt{\frac{1}{2}+\frac{1}{2}A_{j-1}},\ \ B_j = B_{j-1} \times A_j$$

for $j = 1, 2, \ldots$

$A_1 = \sqrt{\frac{1}{2}+\frac{1}{2}\times 0} = \sqrt{\frac{1}{2}}$

$B_1 = B_0 \times A_1 = \sqrt{\frac{1}{2}}$

$A_2 = \sqrt{\frac{1}{2}+\frac{1}{2}\times\sqrt{\frac{1}{2}}}$

$B_2 = B_1 \times A_2$

$\vdots$

```
DATA T;
A=0; B=1;
DO I=1 TO 20;
    A=SQRT(0.5+A/2); B=B*A;
END;
D= 2/B; PUT B= D=;
RUN;
```

The results are

```
B=0.6366197724 D=3.1415926536
```

Example 20

It is known that normal distribution can be used to approximate binomial distribution using the following formula

$$\Pr\left(B(n,p) \le m\right) = \Pr\left(N\left(np, np(1-p)\right) \le m + 0.5\right)$$

Convert discrete to continuous

for $0 \le m \le n$.

Write a SAS program to compute the maximum absolute error of the above approximation with respect to values of m for given n and p.

$$A = \max_{0 \le m \le n} \left|\Pr\left(B(n,p) \le m\right) - \Pr\left(N\left(np, np(1-p)\right) \le m + 0.5\right)\right|$$

```
DATA NORMAL_APPROX;
INPUT N P;
A=0; ← Running maximum
DO K=0 TO N;
A=MAX(ABS(PROBBNML(P,N,K) - PROBNORM((K+0.5-
N*P)/SQRT(N*P*(1-P)))),A);
END;
PUT N= P= A=;
CARDS;
10 0.5
8 0.3
RUN;
```

The results are

```
N=10 P=0.5 A=0.0026861603
N=8  P=0.3 A=0.0210252524
```

When p = 0.5, it is better to be estimated by normal distribution;
When p ≠ 0.5, it is likely to be skewed and not normal distributed.

SAS Input Statements:

1.

```
DATA record;
LENGTH Name $ 13 Address $ 11;
INPUT Name & Address & Sex $ Height Salary COMMA7. Date_of_Employment $ 46-55;

DATALINES;
Chan Tai Man  Sheung Wan  M 168 $31,000      01Jul1993
Lee Siu Ming  Central  F 175 $25,145         01Oct1995
RUN;
```

2.

```
DATA club2;
LENGTH ID 4 Name $ 14 Team $ 6;
INPUT ID Name $ & Team StartWeight EndWeight;

CARDS;
1023 David Shaw  red 189 165
1049 Amelia Serrano   yellow 145 124
1221 Jim Brown  yellow 220 .
RUN;
```

3(a).

```
DATA club1;
INPUT ID 1-4 Name $ 6-12 Team $ 13-19 StartWeight 20-22 EndWeight 24-26;

CARDS;
1023 David  red    189 165
1049 Amelia yellow 145
1246 Ravi   yellow     177
RUN;
```

3(b).

```
DATA club1_subset;
    SET club1;
    IF (NOT(StartWeight EQ . OR EndWeight EQ .)) THEN OUTPUT;
RUN;
```

4(a).

```
DATA Q4a;
LENGTH TIME $ 5 FULL_NAME $ 13;
INPUT TIME $
      FULL_NAME $ 7-19
      +(-13) LAST_NAME $
      PLACE $ 22-35
      SUBJECT $ 37-52
      LENGTH_MEETING $ 54-63
      CONFIRMED $ 68-70
      ;
CARDS;
11:00 Li Lan        Room 30        Personnel review 45 minutes    Yes
13:00 Leung Mei Fai Leung's office Marketing        30 minutes    No
15:00 Mak David     Lab            Test results     20 minutes    Yes
run;
proc print;
run;
```

4(b).

```
DATA Q4b;
LENGTH TIME $ 5 FULL_NAME $ 13 PLACE $ 14 SUBJECT $ 16;
INPUT TIME $
      LAST_NAME $
      @7 FULL_NAME $ &
      PLACE $ &
      SUBJECT $ &
      @52 LENGTH:2.
      CONFIRMED $3.
      ;
CARDS;
11:00 Li Lan       Room 30          Personnel review 45 Yes#
13:00 Leung Mei Fai  Leung's office  Marketing      30 No
15:00 Mak David      Lab            Test results      20 Yes
run;
proc print DATA=Q4b;
    VAR TIME LAST_NAME FULL_NAME PLACE SUBJECT LENGTH CONFIRMED;
run;
```

6(a).

```
DATA Q3a;
* INFILE 'C:\Folder\hotel.txt';
INPUT ROOM 1-3 GUESTS 5
      InM 9 InD 13-14 InY 17-20
      OutM 25 OutD 29-30 OutY 33-36
      RoomType & $13. Rate :DOLLAR8.;

CARDS;
211 3   2   7   2019    2   11   2019 Deluxe Suite    $295
214 2   2   2   2019    2   12   2019 Basic no view  $75
216 4   2   2   2019    2   13   2019 Suite  $255
220 5   2   3   2019    2   12   2019 Basic w/view   $155
221 3   2   3   2019    2   12   2019 Luxury  $195
223 5   2   7   2019    2   13   2019 Suite  $255
238 4   1   31  2019    2   13   2019 Basic w/view   $155
241 1   2   1   2019    2   13   2019 Luxury  $195
run;
PROC PRINT;
title "Hotel";
RUN;
```

6(b).

```
DATA Q3b;
SET Q3a;
InDate = MDY(InM,InD,Iny);
OutDate = MDY(OutM,OutD,Outy);
PROC PRINT;
title "Hotel";
RUN;
```

6(c).

```
DATA Q3c;
SET Q3b;
Charge = (OutDate-InDate)*Rate + 10*GUESTS;
PROC PRINT;
title "Hotel";
RUN;
```

7(a).

```
DATA PERSONNEL;
INPUT ID $ 1-4 DEPT $ 1
      + 5 BIRTHDAY DATE9.
      YEAR 12-15
      + 4 SALARY COMMA8.2;
CARDS;
A123   4Mar1989    8,60000
A037  23Jun1957   21,45000
M015  19Sep1977   17,50000
;
RUN;
```

7(b).

```
DATA PERSONNEL;
INPUT ID: $4.
      +(-5) DEPT $1.
      + 4 BIRTHDAY DATE9.
      +(-4) YEAR 4.
      SALARY COMMA8. /;
CARDS;
A123  4Mar1989  8,6,00
***************
    A037 23Jun1957  21,450
**************
 M015 19Sep1977$17,500
***********
;
RUN;
```

Extra R code:

1.

```
series <- merge(game1, game2, by = c("Initials", "Surname"))[order(-merge(game1, game2, by = c("Initials", "Surname"))$Second.y), ]
```

2.

```
model <- lm(y~x)
c <- coef(model)[1]
m <- coef(model)[2]
```

```
plot(x,y)
abline(c,m)
```

3.

```
my_det <- function(M) {
  if (!is.matrix(M) || nrow(M) != ncol(M)) {
    stop("Input matrix is not a square matrix.")
  }

  if (nrow(M) == 1) {
    return(M[1, 1])
  } else {
    det_value <- 0
    n <- nrow(M)

    for (i in 1:n) {
      sign <- (-1)**(i + 1)
      sub_matrix <- M[-1, -i, drop = FALSE]  # Exclude 1st row & ith column
      det_value <- det_value + sign * M[1, i] * my_det(sub_matrix)
    }
    return(det_value)
  }
}
```

4.

```
bisection<-function(f, x1, x2, n = 1000, err = 1e-05) {
        f1 <-f(x1); f2 <-f(x2)
        if (f1==0) return(x1)
        else if (f2==0) return(x2)
        else if (f1*f2>0) stop("Roots may not exist in range")
        else {
                x <-(x1+x2)/2; fx <-f(x)
                i <-0
                while ((abs(fx)>err)&(i<=n)) {
                        if (fx*f2>0) {
                                x2 <-x
                        } else if (fx*f1>0) {
                                x1 <-x
                        }
                        x <-(x1+x2)/2; fx<-f(x)
                        i <-i+1
                }
        }
        return(x)
}
bisection(fx,0.0001,1,1000,10^-5)
```