Basics of R

Emmanuel S. Tsyawo

10/16/2021

Contents

Creating, indexing, subsetting and operating vectors

Create a vector using concatenation

```
z = 1 # assign a scalar value to z
z # print out value of z to the console
```

```
## [1] 1
```

```
v = c(0,1,7,-2) # create a vector v
v # print vector v to console
```

```
## [1]  0  1  7 -2
```

```
w = c(1,3,-12,0) # create a vector w
w # print vector w to console
```

```
## [1]   1   3 -12   0
```

Arithmetic operations on the vectors and scalars

```
z + v
```

```
## [1]  1  2  8 -1
```

```
w + v
```

```
## [1]  1  4 -5 -2
```

```
w*v
```

```
## [1]   0   3 -84   0
```

```
w/v
```

```
## [1]       Inf  3.000000 -1.714286  0.000000
```

NB: the above operations are element-wise

Indexing elements of a vector

```
w[2] # second element of w
```

```
## [1] 3
```

```
w[10] # ??
```

```
## [1] NA
```

```
length(w)
```

```
## [1] 4
```

It is not possible to access an element beyond the length of the vector.

```
v[1:3] # first 3 elements of v
```

```
## [1] 0 1 7
```

```
v[c(2,4)] # 2nd and 4th arguments of v
```

```
## [1]  1 -2
```

Creating, indexing, subsetting and operating on matrices

Creating and subsetting matrices

Create a 2 x 2 matrix m

```
(m = matrix(c(3,-4.2,-7.1,0.95),nrow=2,ncol=2))
```

```
##      [,1]  [,2]
## [1,]  3.0 -7.10
## [2,] -4.2  0.95
```

What does ( ) around code do? Fill matrix by rows; default is by column

```
(m = matrix(1:6, nrow=2, byrow=TRUE))
```

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

```
m[2,] # Index 2 row
```

```
## [1] 4 5 6
```

```
m[,3] # Index 3rd column
```

```
## [1] 3 6
```

```
m[2,3] #(2,3)'th element of the matrix m
```

```
## [1] 6
```

Matrix Algebra

```
m-2 #subtract 2 from each element in m
```

```
##      [,1] [,2] [,3]
## [1,]   -1    0    1
## [2,]    2    3    4
```

```
m/5 #divide each element in m by 5
```

```
##      [,1] [,2] [,3]
## [1,]  0.2  0.4  0.6
## [2,]  0.8  1.0  1.2
```

NB: these operations are element-wise

```
n=matrix(c(2,4,6,8),ncol = 2) #create a 2x2 matrix

dim(m) #check the dimension of the matrix
```

```
## [1] 2 3
```

```
dim(n) #check the dimension of the matrix
```

```
## [1] 2 2
```

```
n%*%m #matrix product of n and m
```

```
##      [,1] [,2] [,3]
## [1,]   26   34   42
## [2,]   36   48   60
```

```
t(m) # transpose of m
```

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

```
det(n) #determinant of n
```

```
## [1] -8
```

```
solve(n) #matrix inverse of n
```

```
##      [,1]  [,2]
## [1,] -1.0  0.75
## [2,]  0.5 -0.25
```

```
eigen(n) #eigen-value decomposition of n
```

```
## eigen() decomposition
## $values
## [1] 10.7445626 -0.7445626
##
## $vectors
##            [,1]       [,2]
## [1,] -0.5657675 -0.9093767
## [2,] -0.8245648  0.4159736
```

Creating, indexing, subsetting and operating lists

Lists are more flexible (than vectors and matrices) for storing objects in R. Lists can store objects of different types, e.g, characters, integers, real numbers, complex numbers, etc.

```
s = c("Kofi","Kojo","Ziggy") #create a vector of strings
s
```

```
## [1] "Kofi"  "Kojo"  "Ziggy"
```

```
L = list(sc=z,v1=v,v2=w,m=m,n=n,s=s)
L
```

```
## $sc
## [1] 1
##
## $v1
## [1]  0  1  7 -2
##
## $v2
## [1]   1   3 -12   0
##
## $m
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
##
## $n
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
##
## $s
## [1] "Kofi"  "Kojo"  "Ziggy"
```

```r
length(L) #check number of elements in list L
```

```
## [1] 6
```

```r
L$sc #access the element sc in L
```

```
## [1] 1
```

```r
L[[1]] #access the first element in L
```

```
## [1] 1
```

Loading and manipulating data in R

One way to go about loading a data set into R is to first set a working directory.

To set working directory to the source file folder in RStudio: use the steps

Session -> Set working directory -> To Source file location

Loading a csv data file

Let us load the .csv data from the working directory. This is the mroz dataset taken from the Jeffery Wooldridge's textbook website. Ensure the data set is in the working directory.

```r
dat<- read.csv("dat.csv",header = T,sep = " ")
names(dat) #variable names
```

```
## [1] "y"          "age"        "education"  "experience" "nonwife"
```

```r
dim(dat) # dimension of the dataframe
```

```
## [1] 753   5
```

```r
nr = dim(dat)[[1]] # extract number of rows
nc = dim(dat)[[2]] # extract number of columns
nr
```

```
## [1] 753
```

```r
nc
```

```
## [1] 5
```

View a summary of the data set

```r
summary(dat) #summarise the data
```

```
##        y               age          education       experience
##  Min.   :0.0000   Min.   :30.00   Min.   : 5.00   Min.   : 0.00
##  1st Qu.:0.0000   1st Qu.:36.00   1st Qu.:12.00   1st Qu.: 4.00
##  Median :1.0000   Median :43.00   Median :12.00   Median : 9.00
##  Mean   :0.5684   Mean   :42.54   Mean   :12.29   Mean   :10.63
##  3rd Qu.:1.0000   3rd Qu.:49.00   3rd Qu.:13.00   3rd Qu.:15.00
##  Max.   :1.0000   Max.   :60.00   Max.   :17.00   Max.   :45.00
##     nonwife
##  Min.   :-0.02906
##  1st Qu.:13.02504
##  Median :17.70000
##  Mean   :20.12896
##  3rd Qu.:24.46600
##  Max.   :96.00000
```

```
xx = as.matrix(cbind(dat$age,dat$experience)) #create a design matrix of two variables
```

Split the data set into two, even and odd-indexed rows

```
eI = (1:floor(nr/2))*2 # even indices
eDat<- cbind(dat$y[eI],xx[eI,]) # subset even-indexed observations of y and xx
oDat<- cbind(dat$y[-eI],xx[-eI,]) # odd-indexed observations of y and xx
```

NB: The negation of an index is all but those observations.

Summarise both subsets of data

```
summary(eDat)
```

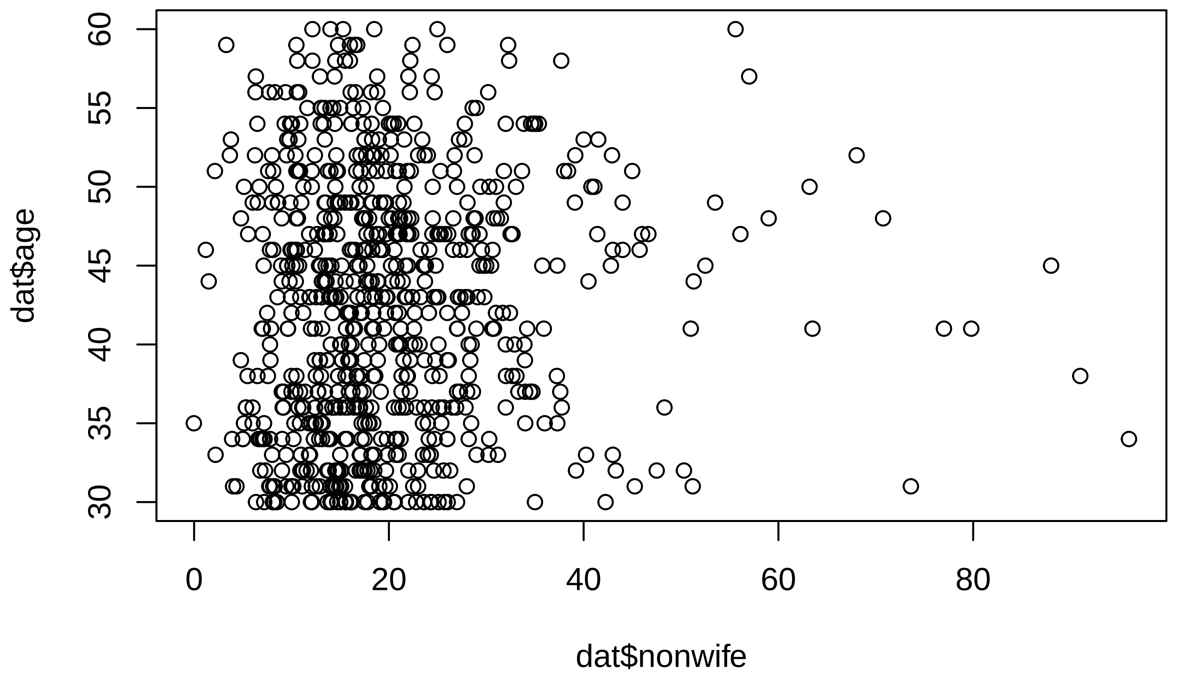
```
##        V1               V2              V3
##  Min.   :0.0000   Min.   :30.00   Min.   : 0.0
##  1st Qu.:0.0000   1st Qu.:35.75   1st Qu.: 4.0
##  Median :1.0000   Median :42.00   Median : 9.0
##  Mean   :0.5691   Mean   :42.43   Mean   :10.4
##  3rd Qu.:1.0000   3rd Qu.:49.00   3rd Qu.:14.0
##  Max.   :1.0000   Max.   :60.00   Max.   :45.0
```

```
summary(oDat)
```

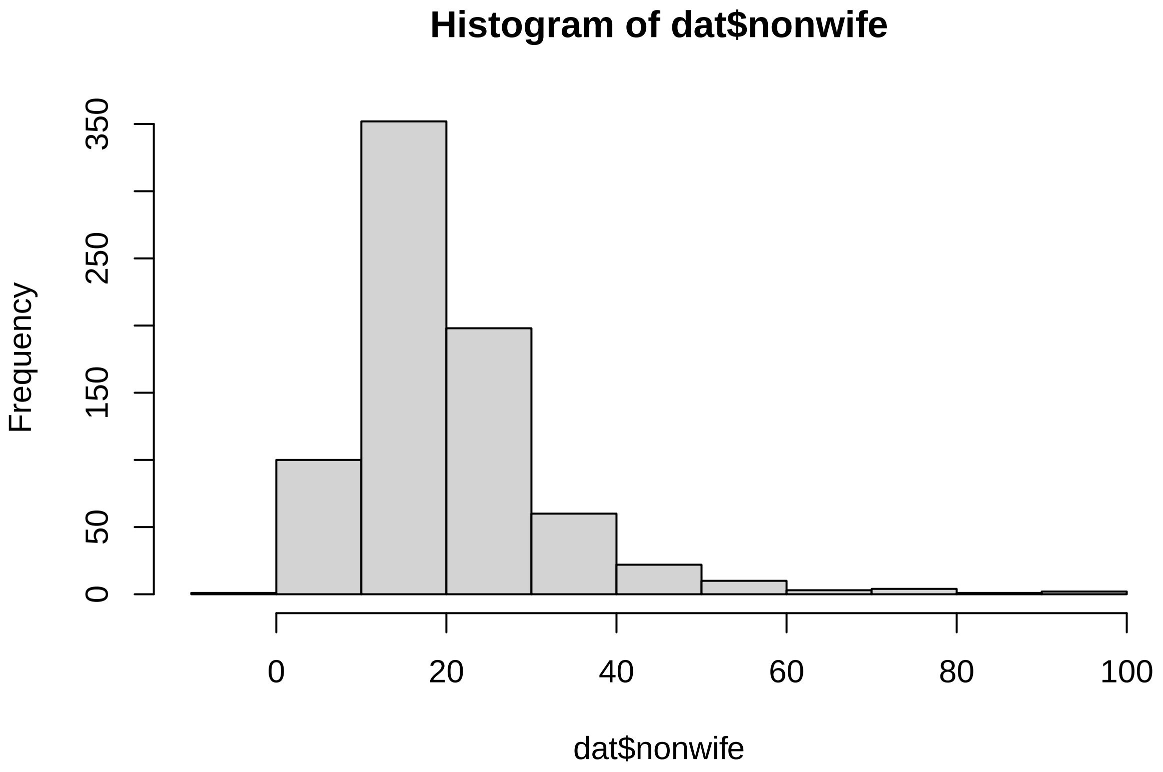
```
##        V1               V2              V3
##  Min.   :0.0000   Min.   :30.00   Min.   : 0.00
##  1st Qu.:0.0000   1st Qu.:36.00   1st Qu.: 4.00
##  Median :1.0000   Median :43.00   Median : 9.00
##  Mean   :0.5676   Mean   :42.65   Mean   :10.86
##  3rd Qu.:1.0000   3rd Qu.:48.00   3rd Qu.:15.00
##  Max.   :1.0000   Max.   :60.00   Max.   :38.00
```

Plotting data in R

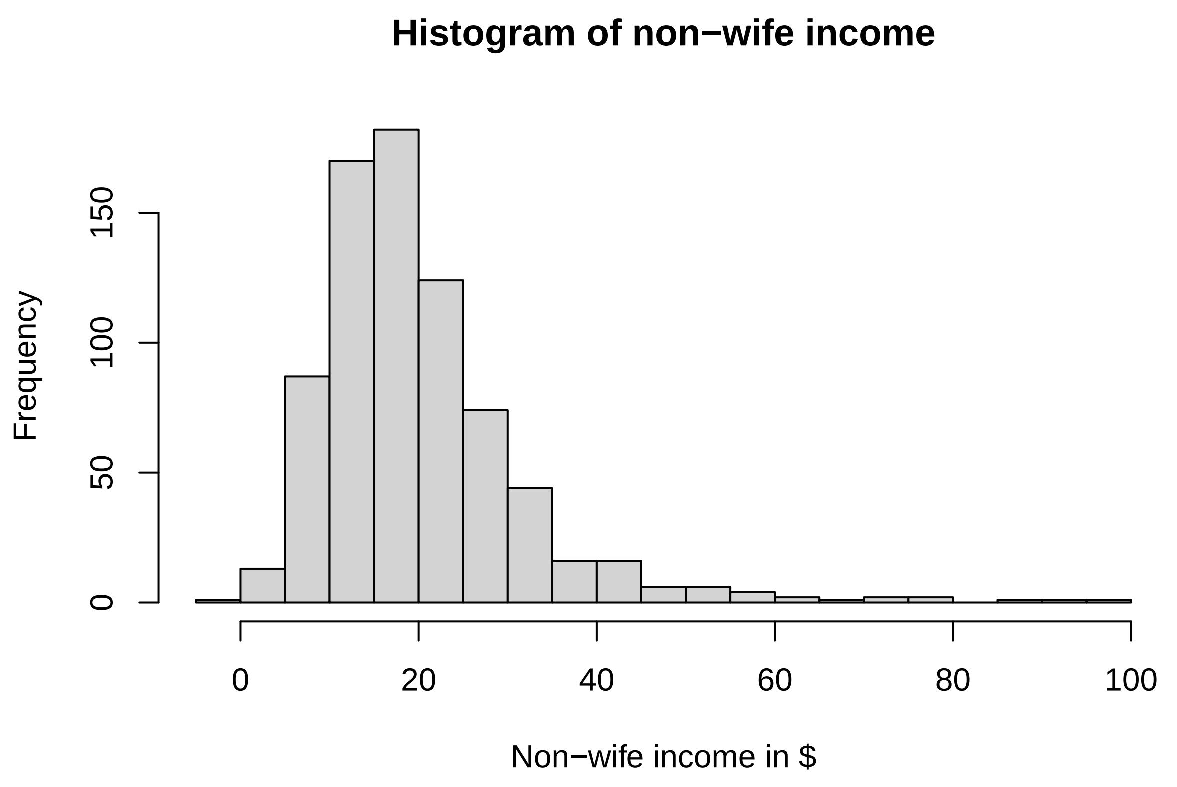
```
plot(dat$nonwife,dat$age) # a scatter plot of the variable nonwife and age
```



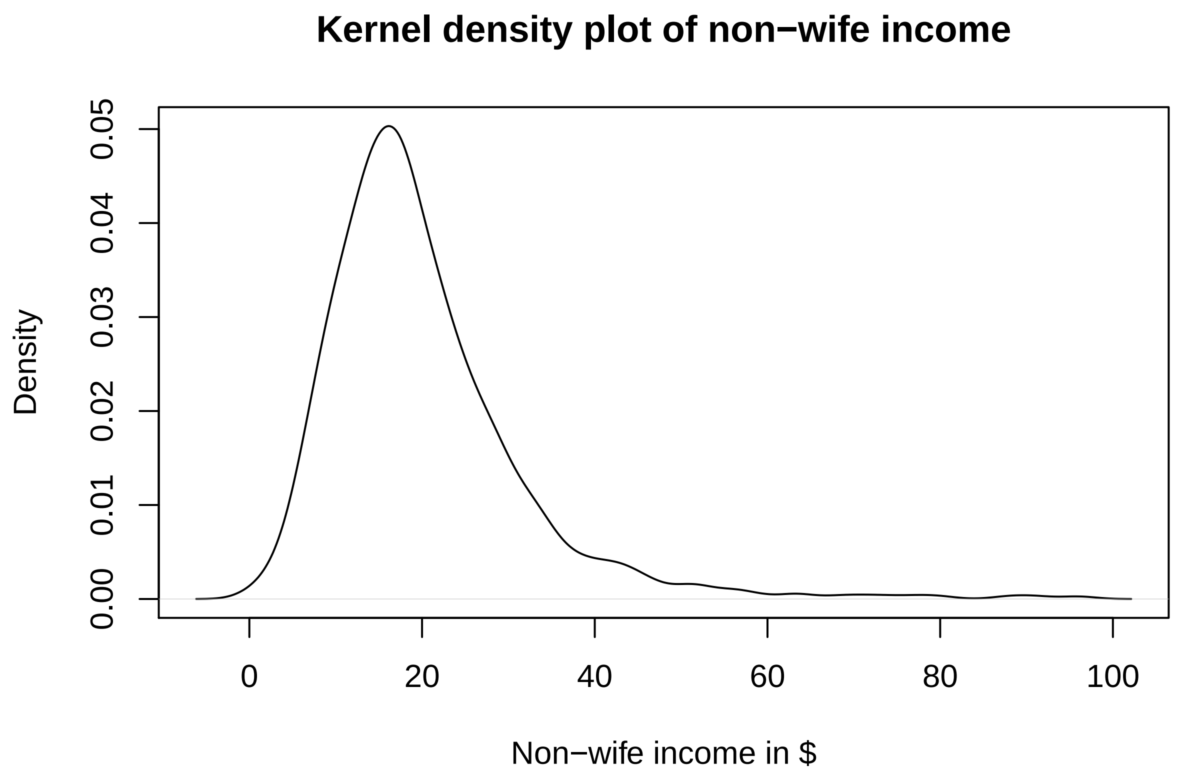
```r
hist(dat$nonwife) # plot histogram
```



```r
hist(dat$nonwife, breaks=30, main = "Histogram of non-wife income",
     xlab = "Non-wife income in $") #a histogram with 30 bins
```



```
plot(density(dat$nonwife),main = "Kernel density plot of non-wife income",
     xlab = "Non-wife income in $") #a kernel density plot
```



Logicals in R

Logicals are useful mainly for verifying whether statements are true or false. These are then used in writing functions and algorithms.

Examples:

Verify equalities and inequalities

```
2 == 3 # is 2 equal to 3? # Note == is logical, a=b assigns value b to a
```

```
## [1] FALSE
```

```
2 != 3 # is 2 not equal to 3?
```

```
## [1] TRUE
```

```
2<3 # is 2 less than 3?
```

```
## [1] TRUE
```

```
2>=3 # is 2 greater or equal to 3?
```

```
## [1] FALSE
```

Using logical statements in subsetting vectors and matrices

```
which(w==0) # which element(s) of vector w equals 0?
```

```
## [1] 4
```

```
w[which(w==0)] # extract such element in w
```

```
## [1] 0
```

```
which(v==12) # which element of vector v equals 12?
```

```
## integer(0)
```

```
which(w%%2==0) #indices of even numbers in w i.e the modulo of which numbers =0?
```

```
## [1] 3 4
```

```
w[which(w%%2==0)] #even numbers in w
```

```
## [1] -12   0
```

```
w[-which(w%%2==0)] #non-even numbers in w
```

```
## [1] 1 3
```

```
#or
w[-which(w%%2!=0)]
```

```
## [1] -12   0
```

Verify joint statements

```
any(w< -1) # any element of w less than -1? NB. ensure space between < and -
```

```
## [1] TRUE
```

```
w %in% v # which elements of w are in v?
```

```
## [1]  TRUE FALSE FALSE  TRUE
```

```
all(w==v) # are vectors w and v exactly equal, i.e. element-wise?
```

```
## [1] FALSE
```

```
w==v # check element-wise equality
```

```
## [1] FALSE FALSE FALSE FALSE
```

if/else statements

These statements enable us to carry out a task only if conditions are satisfied.

```
i=1
if(w[i]<0){
  print(paste(w[i],"is a negative number"))
}else if(w[i]>0){
  print(paste(w[i],"is a positive number"))
}else{
  print(paste(w[i],"is neither positive nor negative"))
}
```

```
## [1] "1 is a positive number"
```

Change the value of i to other indices and see what happens

The following programme prints whether a number is odd or even.

```
i=3
if(w[i]%%2==0){
  print(paste(w[i],"is an even number"))
}else{
  print(paste(w[i],"is an odd number"))
}
```

```
## [1] "-12 is an even number"
```

A compact usage with the `ifelse()` command

```
set.seed(333) #set seeed for reproducibility
x = round(rnorm(10),2) #randomly sample 10 values from the standard normal distribution
y = ifelse(x>0, 1, -1) #y is 1 if x is positive, -1 otherwise
rbind(x,y) #view a vertical concatenation of x and y
```

```
##    [,1] [,2]  [,3] [,4]  [,5]  [,6] [,7] [,8] [,9] [,10]
## x -0.08 1.93 -2.05 0.28 -1.53 -0.27 1.23 0.63 0.35 -0.56
## y -1.00 1.00 -1.00 1.00 -1.00 -1.00 1.00 1.00 1.00 -1.00
```

Loops

Loops enable a repetition of steps for a given number of times or until some condition is met.

for loop:

This type of loop is suitable for a finite number of steps known before hand. For example, summing up numbers sequentially:

```r
sum = 0 #initialise sum to zero
for (i in 1:10) sum = sum + i
sum
```

```
## [1] 55
```

```r
sum(1:10)
```

```
## [1] 55
```

Taking products of scalars sequentially

```r
pr = 1
for(j in 1:10) pr = pr*j
pr
```

```
## [1] 3628800
```

A slighly more complicated example: let

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}.$$

Now consider the matrix powers of $\mathbf{A}$, i.e., $\mathbf{A}^n$

Taking the powers of $\mathbf{A}$ sequentially. Store them in a list

```r
A = matrix(c(1,0,-1,1),ncol = 2)
print(A) #view A
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    0    1
```

```r
An.List=list() #initialise an empty list to store matrix powers
An.List[[1]]=A #initialise list with A
for(j in 2:5) An.List[[j]]=An.List[[j-1]]%*%A
An.List
```

```
## [[1]]
##      [,1] [,2]
## [1,]    1   -1
## [2,]    0    1
##
## [[2]]
##      [,1] [,2]
## [1,]    1   -2
## [2,]    0    1
##
## [[3]]
##      [,1] [,2]
## [1,]    1   -3
## [2,]    0    1
##
## [[4]]
##      [,1] [,2]
## [1,]    1   -4
## [2,]    0    1
##
## [[5]]
##      [,1] [,2]
## [1,]    1   -5
## [2,]    0    1
```

For a slightly more complicated example, sum over only even numbers:

```
sum = 0
for (i in 1:10){
  if (i%%2 == 0) sum = sum + i
}#end for loop
print(sum)
```

```
## [1] 30
```

The use of curly brackets for a loop is advisable if you have several lines to execute in a loop.

Exercise: sum over the odd numbers from 1 through 20

while loop:

This type of loop is suitable for a known stopping criterion but not the number of steps.

Example: Use a while loop to report the (random) number of steps it takes to get from zero to a number greater than 10 given random increments $x_i - x_{i-1} \sim \mathcal{N}(0.5, 1)$, i.e., the normal distribution with mean 0.5 and standard deviation 1.

```
x=0
n=0
set.seed(333)
# set seed when using random number generation for reproducibility of results
while(x <= 10) {
  n=n+1
  x=x+rnorm(1,mean=.5,sd=1)
  }#end while loop
print(paste ("n = ", n, ", x = ",round(x,2) )) #print out results
```

```
## [1] "n =  26 , x =  11.05"
```

Exercise: Use a while loop to search on the interval [-2,4] for the maximum of $f(x) = -(x-1)^2$ using increments of 0.001.

Solution:

```
df = 10 #set a dummy to a positive number
x = -2 #searching from left to right
#all increments in the function till the maximum point should be positive.
fx1 = -(x-1)^2
n = 0 #initialise the number of iterations
while(df>0){
  n=n+1
  x = x + 0.001
  fx2 = -(x-1)^2
  df = fx2-fx1 #compute change in function value
  fx1=fx2 #update function value
}#end while loop
x-0.001 #report the maximiser
```

```
## [1] 1
```

```r
n-1 #report number of iterations
```

```
## [1] 3000
```

User defined functions in R

Functions in R are key for executing tasks in a neat and orderly way. These make them reusable. They take input and provide output. The general form of a function definition is

```
f = function(x,y,...){
expression involving inputs x, y, ...
return(function value)
}
```

The result of the function will be the last evaluated expression, unless return(function value) is used.

Simple functions

Here is a simple function that computes the power of a matrix using the for-loop example.

```r
mat.pow = function(A,n) {
  An=A #initialise An to A
  for(j in 2:n) An=An%*%A
  An
}
mat.pow(A,5)
```

```
##      [,1] [,2]
## [1,]    1   -5
## [2,]    0    1
```

Consider this simple Cobb-Douglas production function: $Q(K, L) = K^{0.4}L^{0.6}$.

```r
Q = function(K,L) {(K^0.4) * (L^0.6)}
Q(200,40)
```

```
## [1] 76.14616
```

Vary inputs and verify output

Exercise: Code the following function: $f(x) = \exp(-(x-1)^2)$. Plot it using the `curve()` function over the interval [-2,4].

Ordinary Least Squares

Example: A complicated example - a function to compute OLS results. Write a function to compute OLS estimates $\hat{\beta} = (X'X)^{-1}X'Y$, standard errors from the homoskedastic covariance matrix $\hat{V}_\beta = \hat{\sigma}^2(X'X)^{-1}$ with $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^n \hat{U}_i^2$, $\hat{U}_i = Y_i - X_i\hat{\beta}$, t-statistics, and p-values. Recall the p-value from a t-statistic t_n is $P(|T| > |t_n|) = 1 - P(|T| \le |t_n|) = 2P(|T| \le -|t_n|)$.

```r
OLS<- function(Y,X){
  Xvar.names=names(X)
  N = length(Y) #obtain number of observations
  X = as.matrix(cbind(1,X)) # include 1's for the intercept term
  k=ncol(X) # number of parameters to estimate
  beta = solve(t(X)%*%X)%*%t(X)%*%Y # compute the k x 1 vector of beta_hat
  U = Y - X%*%beta # compute residuals
  df = N - k  #degree of freedom
  sig2 = sum(U^2)/N #compute sigma squared
  varcov<- sig2*solve(t(X)%*%X)*N/(df) # compute the homoskedastic covariance matrix
  m = matrix(NA,nrow = 4,ncol = k) # a matrix to store regresion results
  m[c(1,2),] = rbind(t(beta),sqrt(diag(varcov))) # first two rows to store parameters and standard erro
  t.stat = m[1,]/m[2,] # compute t statistics
  pval = 2*(pt(-abs(t.stat),df)) #compute p values taken from the t distribution
  m[c(3,4), ] <- rbind(t.stat,pval) # store t-stats and p-values in 3rd and 4th rows
  dimnames(m)[[1]]<- c("estimate", "std. error","t stat","p value")
  dimnames(m)[[2]]<- c("Intercept",Xvar.names)
  # label the rows
  return(t(m))
}
```

Example:

```
reg<- OLS(Y=dat$nonwife,X=dat[c("age","education","experience")])
reg
```

```
##               estimate std. error    t stat      p value
## Intercept  -6.9255329 3.22124829 -2.149953 3.187847e-02
## age         0.2614656 0.05235361  4.994223 7.355649e-07
## education   1.6132605 0.17508333  9.214244 3.101009e-19
## experience -0.3658783 0.05211057 -7.021190 4.939214e-12
```

Compare to the internal `lm()` R function

```
regI<- lm(nonwife~age+education+experience,data = dat)
summary(regI)
```

```
##
## Call:
## lm(formula = nonwife ~ age + education + experience, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.506  -6.623  -1.827   3.801  66.976
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.92553    3.22125  -2.150   0.0319 *
## age          0.26147    0.05235   4.994 7.36e-07 ***
## education    1.61326    0.17508   9.214  < 2e-16 ***
## experience  -0.36588    0.05211  -7.021 4.94e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.8 on 749 degrees of freedom
## Multiple R-squared:  0.1421, Adjusted R-squared:  0.1387
## F-statistic: 41.35 on 3 and 749 DF,  p-value: < 2.2e-16
```

Compare both results

Log-likelihood for the normal linear model

Assumption $U_i \sim \mathcal{N}(0, \sigma^2)$ where $U_i = Y_i - X_i\beta$ and data are *i.i.d.*

Recall the likelihood function is $L_n(\beta,\sigma^2) = \prod_{i=1}^n \phi(U_i(\beta)) = \frac{1}{(2\pi)^{n/2}} \exp(-\sum_{i=1}^n (Y_i - X_i\beta)^2/(2\sigma^2))$. The log-likelihood is $\ell(\beta,\sigma^2) = \log L_n(\beta,\sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - X_i\beta)^2$.

```
llike_lnorm<- function(pars,Y,X){ #pars is the vector [beta,sigma]
  N = length(Y) #obtain number of observations
  X = as.matrix(cbind(1,X)) # include 1's for the intercept term
  k=ncol(X) # number of parameters to estimate
  np = length(pars) #obtain number of parameters (including sigma)
  if(np!=(k+1)){stop("Not enough parameters.")} #ensure the number of parameters is correct
  beta = matrix(pars[-np],ncol = 1) #obtain column vector of slope parameters beta
  sig2 = pars[np] #sigma^2
  U<- Y - X%*%beta #compute U
  ll = -(N/2)*log(2*pi*sig2) -sum(U^2)/(2*sig2)#obtain log joint likelihood
  return(ll)
}
# example: #return log likelihood value for parameter values of 1's
llike_lnorm(pars = rep(1,5),Y=dat$nonwife,X=dat[c("age","education","experience")])
```

```
## [1] -928647.2
```

```
llike_lnorm(pars = c(regI$coefficients,10),Y=dat$nonwife,X=dat[c("age","education","experience")])
```

```
##
## -5925.516
```

Writing a piecewise function

Take the simple example

$$f(x) = \begin{cases} -x^4 & x < 0 \\ 0 & 0 \le x \le 2 \\ (x-2)^3 & x > 2 \end{cases}$$

Write the function and plot it over the interval $[-4, 10]$.

```
piecefn<- function(x){
  if(x<0){
    y=-x^4
  }else if(x>2){
    y=(x-2)^3
  }else{
  y = 0
  }
  return(y)
}
```

Always test your function

```
piecefn(-3)
```

```
## [1] -81
```

```
piecefn(1)
```

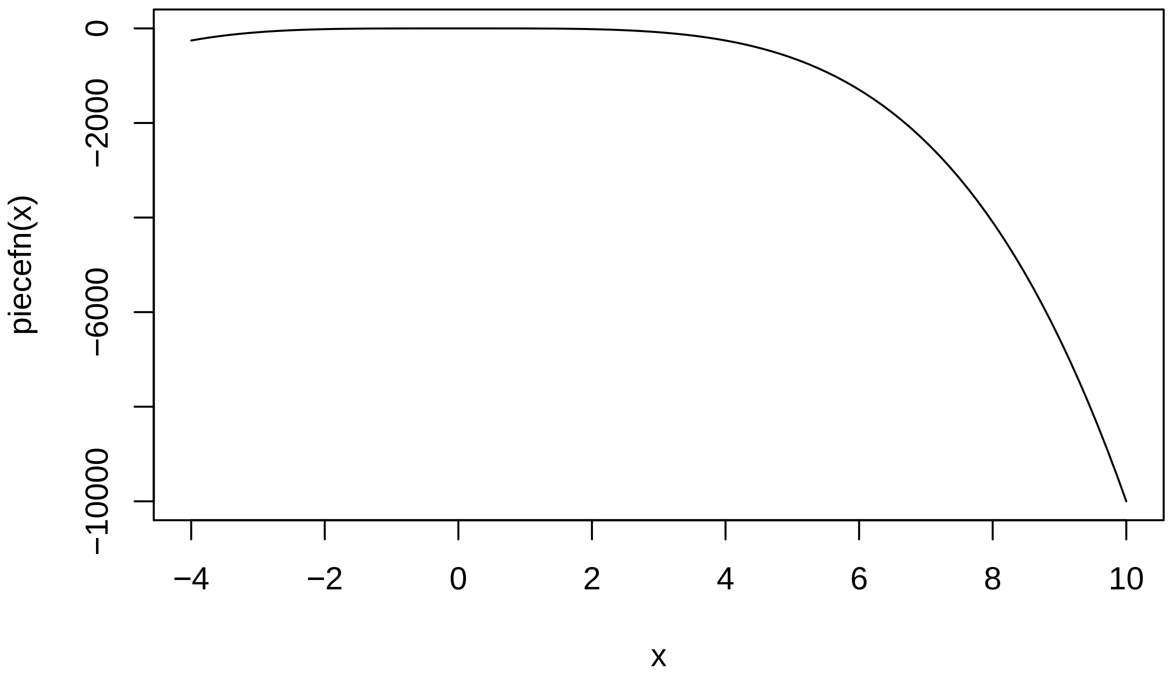
```
## [1] 0
```

```
piecefn(3)
```

```
## [1] 1
```

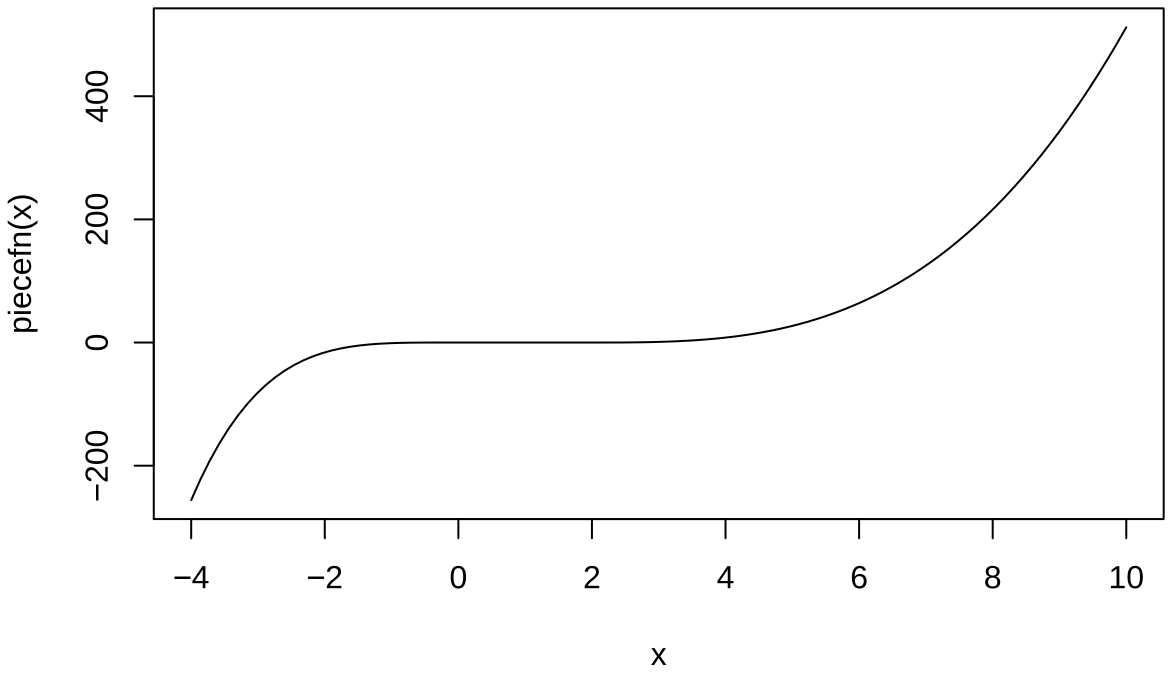
```r
curve(piecefn,from = -4,to=10) #attempt plotting the curve
```

```
## Warning in if (x < 0) {: the condition has length > 1 and only the first element
## will be used
```



Some functions need to be vectorised for plotting and in general for operating on vector inputs.

```r
piecefn=Vectorize(piecefn) # vectorize the function. why?
curve(piecefn,from = -4,to=10) #plot the curve
```



Generating random numbers in R

Sometimes, we may want to obtain draws from a distribution or randomise certain operations. This can be done in a number of ways.

Example: Sample 10 values from the standard uniform distribution $\mathcal{U}[0,1]$.

```
(x = runif(10))
```

```
##  [1] 0.91258126 0.64221472 0.96317027 0.15904822 0.65935954 0.96105376
##  [7] 0.01434009 0.82467439 0.24109348 0.38955671
```

Repeat the above step a number of times. Are the numbers the same in each draw? Now set seed to any integer, say 40

```
set.seed(40) ; (x = runif(10))
```

```
##  [1] 0.6835820 0.8729038 0.6901173 0.1159361 0.1950091 0.4612009 0.2035352
##  [8] 0.5908492 0.3738881 0.1412981
```

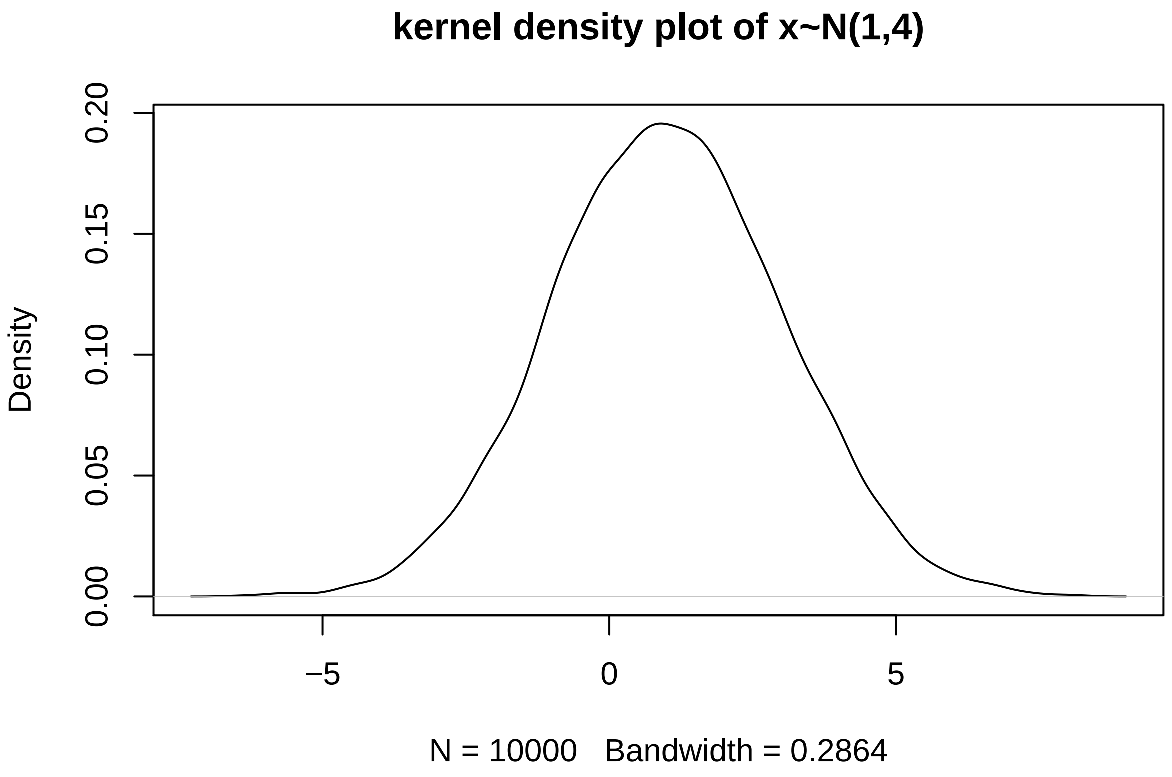
Repeat the above steps a number of times. What do you observe?

Example: Make 10 000 draws from the distribution $\mathcal{N}(1,4)$, i.e., mean 1, standard deviation 2

```
set.seed(40) ; x = rnorm(10000,mean=1,sd=2)
```

Make a density plot

```
plot(density(x), main = "kernel density plot of x~N(1,4)")
```



Exercises:

Let $X \sim \mathcal{N}(0,1)$. Simulate 10 000 samples from the distribution of X. Set seed at 10.

1. Use the random samples to compute the first four central moments of X.
2. Use the random samples to compute $\text{cov}(X, |X|)$.
3. Using the random samples, compute $\mathbb{P}(|X| > 1)$.
4. Consider the same linear model $U_i = Y_i - X_i\beta$ with a different assumption $U_i \sim \lambda(0, 1)$, i.e., the logistic distribution with location parameter 0 and scale parameter 1. Recall the probability density function of the logistic distribution is $f(x) = \frac{\exp(-x)}{(1+\exp(-x))^2}$.
   a. Derive the log-likelihood function $\ell(\beta)$.
   b. Write a function that takes as input β, Y, and X.
5. Repeat question 4. above for $Cauchy(0, 1)$, i.e., the Cauchy distribution with location 0 and scale parameter 1. Recall the pdf is $f(x) = \frac{1}{\pi(1+x^2)}$.