EJERCICIOS DE APOYO PARA R

Este documento ha sido realizado y revisado por alumnos de primero del grado en Biotecnología de la Universidad Politécnica de Madrid. Se basa en los contenidos de la asignatura de Fundamentos de Programación, impartida por Arturo Hidalgo López y Ángel Fidalgo Blanco.

El **lenguaje** de programación utilizado es **R**, enfocado al análisis estadístico y muy utilizado en investigación científica y biomédica, bioinformática, aprendizaje automático (*machine learning*), minería de datos, y matemáticas financieras.

En este apartado se propondrán algunos ejercicios, que pueden tener una mayor relevancia en sus respectivas prácticas. Con lo que es recomendable que se vayan realizando acorde con la práctica en la que uno se encuentre.

PRÁCTICA 1 Y 2:

1.- Relación de variables con datos dados:

PASOS A SEGUIR

• Crear los objetos pais1, pais2, poblacion1, poblacion2 de manera que pais1 contenga el valor Italia, pais2 contenga el valor Alemania, poblacion1 contenga el valor 50, poblacion2 contenga el valor 80.

• listar todo lo que hay en memoria.

• listar objetos que contengan el carácter p.

• listar objetos que contengan el carácter b.

• listar objetos que comiencen con el carácter b.

• listar objetos que comiencen con el carácter p

• listar todos los objetos que hay en memoria de manera que se obtengan sus características.

• borrar los objetos que empiecen pais1, poblacion1.

• volver a listar el contenido de la memoria.

RAZONAMIENTO

1. Como ya se sabe, las variables pueden contener valores o nombres (entre comillas). Por lo que se procede a asignar los nombres y valores a las variables que se nos proponen.
2. Para listar todos las variables que hayamos almacenado, usaremos el comando ls(), con el que todos se mostrarán.
   NOTA: Si se pidiera mostrar un único valor, se usaría el comando print(), con la variable correspondiente.
3. Para que muestre las variables que contengan la letra p, usaremos la instrucción pat="_", en la que se mostrarán todas aquellas variables que contengan la letra que hayamos puesto entre comillas, habiendo puesto esta condición en los paréntesis del ls().

4. En el caso de que sea el comienzo de la variable la que queremos encontrar, escribiremos pat="^_".
5. Para mostrar todas las variables con sus respectivos valores o nombres, escribiremos ls.str().
6. Cuando queramos eliminar alguna variable, con usar el comando rm(), será suficiente y conseguiremos que las variables que hayamos introducido en estos paréntesis, sean borrados de la memoria.

SOLUCIÓN

```
#1.Crear las variables
pais1="Italia"
pais2="Alemania"
poblacion1=50
poblacion2=80
#2.Listar todo lo que hay en la memoria
ls()
#3.Listar lo que contenga la letra "p"
ls(pat="p")
#4.Listar lo que contenga la letra"b"
ls(pat="b")
#5.Listar lo que comience por la letra "b"
ls(pat="^b")
#6.Listar lo que empiece por la letra "p"
ls(pat="^p")
#7.Listar todos los objetos con sus características
ls.str()
#8.Borrar de la memoria pais1 y poblacion1
rm(pais1,poblacion1)
#9.Volver a listar el contenido de la memoria
ls()
```

2.- Operaciones con vectores y matrices

PASOS A SEGUIR

- Definir los vectores u y v y las matrices A y B

$$u = \begin{pmatrix} -10 & e & \sin(2\pi/3) & -5 & \frac{4}{3} \end{pmatrix}$$

$$v = \begin{pmatrix} 3 & 25 & 10^3 & -50 & \sqrt{13} \end{pmatrix}$$

$$A = \begin{pmatrix} \frac{5}{3} & e^2 & \cos\left(\frac{3\pi}{5}\right) & 10 \\ 10^2 & 0.5 & -2 & \sqrt{6} \\ 88 & 99 & 0.25 & \log(13) \\ 4! & 3 & 0 & -17 \end{pmatrix}$$

$$B = \begin{pmatrix} -6 & 12 & \cos(3) & \tan(8) \\ -\frac{3}{8} & 1 & 1 & 88 \\ \sqrt{88} & \pi & -51 & \log(6) \\ 8! & 2 & -1 & 14 \end{pmatrix}$$

- sumar los vectores u, v
- restar los vectores u, v
- realizar el producto escalar de los vectores u, v
- construir una matriz C cuyas columnas sean los vectores u, v

- construir una matriz R cuyas filas sean los vectores u, v
- sumar las matrices A y B
- multiplicar las matrices A y B
- resolver el sistema de ecuaciones Ax=z, siendo z el vector

$$z = \begin{pmatrix} 2 & 1 & -1 & 3 \end{pmatrix}$$

1. obtener la inversa de la matriz A
2. obtener la transpuesta de la matriz B y almacenarla en BT
3. Realizar el producto de las matrices A y BT
4. Obtener los valores propios de la matriz BT

RAZONAMIENTO

1. Para definir los vectores, escribimos todos sus valores en el comando c().
2. En el caso de las matrices, un procedimiento que podemos seguir, es definir antes con vectores cada una de las filas de las matrices (por separado) y con cuidado de no dejar suelto ningún valor y atendiendo a uso de comas y puntos en los números decimales. Una vez definidos estos vectores, usamos el comando rbind() y ponemos todos los vectores que conformen las columnas de nuestra matriz.
   NOTA: Si se quisiera se podrían definir las columnas, pero después se escribiría el cbind().
   NOTA: rbind se refiere a row='filas'; mientras que cbind, columna='columna'.
3. Para sumar y restar vectores, se usan las reglas matemáticas básicas.
4. Para hacer el producto escalar, se puede hacer de varias formas, pero la más sencilla es escribir la operación _%*%_
5. Al igual que los vectores, las matrices se usan y se restan con las reglas básicas.
6. En el caso del producto de matrices, podemos ver su símil en el producto escalar de vectores, ya que se usa el mismo comando _%*%_
7. Para las distintas aplicaciones y transformaciones de una matriz, usamos los comandos que correspondan a dicha transformación poniendo entre paréntesis la matriz.

SOLUCIÓN

```
#1.Definir los vectores u y v
u=c(-10,exp(1),sin(2*pi/3),-5,4/3)
v=c(3,25,10^3,-50,13^(1/2))
#2.Definir las matrices A y B
a1=c(5/3,exp(2),cos(3*pi/5),10)
a2=c(10^2,0.5,-2,6^0.5)
a3=c(88,99,.25,log(13))
a4=c(factorial(4),3,0,-17)
A=rbind(a1,a2,a3,a4)
b1=c(-6,12,cos(3),tan(8))
b2=c(-3/8,1,1,88)
b3=c(88^.5,pi,-51,log(6))
b4=c(factorial(8),2,-1,14)
B=rbind(b1,b2,b3,b4)
#3.Sumar u y v
u+v
#4.Restar u y v
u-v
```

```
#5.Producto escalar de u y v
u%*%v
#6.Matriz C
C=cbind(u,v)
#7.Matriz R
R=rbind(u,v)
#8.Sumar A y B
A+B
#9.Multiplicar A y B
A%*%B
#10.Inversa de A
solve(A)
#11.Traspuesta de B
BT=t(B)
#12.Producto de A y BT
A%*%BT
#13.Valores propios de BT
BT
```

3.- Representación gráfica de funciones

PASOS A SEGUIR

2. Representa gráficamente la función f(x) = 20cos(5x)e −x para x ∈ [0, 10], empleando 80 valores de x en dicho intervalo y etiquetando el eje de abscisas con el nombre: tiempo y el eje de ordenadas con el nombre: Oscilación. El color será verde. El título del gráfico será: Oscilación amortiguada.

3. Representa gráficamente las funciones sin(x) y cos(x) x ∈ [0, 2π] en un mismo gráfico. La primera de ellas en azul oscuro y la segunda en rojo oscuro.

RAZONAMIENTO

1. En un primer momento, tenemos que obtener los valores para los que queremos que la función se dibuje, además de poner los límites en los que queremos que se encuentren esos puntos. Para ello, usamos el comando x=seq(), en el que se calcularán una serie de valores equidistantes entre sí. Dentro del paréntesis escribimos los límites, y en el tercer término del paréntesis escribimos length=_ para decir al programa cuántos valores debe tomar.
2. Una vez calculados los valores que tomará la x, escribimos la función que queremos representar.
3. Para dibujarla, usamos el comando plot(), en el que escribimos en los dos primeros términos la x y la y (que en este caso es f; f(x)=y). En los siguientes términos, escribimos los nombres que le daremos a los ejes coordenados (xlab, ylab), el color de nuestra gráfica (col), y el título que tendrá (main).
4. Si se quieren representar varias gráficas juntas, se escribe par(new=”TRUE”) entre ambas funciones, y se pondría en las sucesivas funciones si quisiéramos dibujar más gráficas.
   NOTA: No es obligatoria, pero sería recomendable dar distintos colores a las gráficas para que se diferencien con facilidad.

SOLUCIÓN

```
#2.Representar f(x)
x=seq(0,10,length=80)
f=20*cos(5*x)*exp(1)^(-x)
plot(x,f,xlab="Tiempo",ylab="Oscilación",col="green",main="Oscilación
amortiguada",type="b")
#3.Representar sin(y) y cos(y)
y=seq(0,2*pi,length=20)
g=sin(y)
h=cos(y)
plot(y,g,col="dark blue")
par(new=TRUE)
plot(y,h,col="dark red")
```

PRÁCTICA 3:

1.- Operaciones con vectores usando bucles y construcción de tablas

PASOS A SEGUIR

Dados los vectores: v=(12,-3,5,18.7) y w=(12,0.25,77,exp(2))

1) Obtener la suma de los dos vectores mediante bucles y comprobar empleando v+w.

2) Obtener la suma de las componentes del vector v y almacenarlos en SumaC.

3) Realizar el producto escalar de ambos vectores y comprobar empleando %*%.

4) Multiplicar ambos vectores componente a componente.

5) Realizar la operación: zj=vj+2wj , j = 1,…,length(v).

6) Construir una tabla, data.frame que contenga:

'Suma'	*Resultado del apartado 2)*
'Producto escalar'	*Resultado del apartado 3)*

RAZONAMIENTO

1. Partiendo de que ya sabemos escribir vectores, para el bucle debemos ponerle unos límites, en este caso (y en la mayoría de los casos será así) nuestra variable del bucle i varía desde 1 hasta n. Por lo que tenemos que definir n como la longitud de cualquiera de los dos vectores.
2. Para evitar posibles errores, el vector z (suma de v y w) lo iniciaremos a 0, mediante z=c(0).
3. A partir de este momento, podemos crear el bucle para que cada componente del vector z se calcule como suma de las componentes de v y w, una a una.
   NOTA: No es obligatorio, pero para mantener un orden y ver mejor donde acaban y empiezan nuestros bucles, es bueno acostumbrarse a usar las sangrías que desplazan

las líneas de texto hacia la derecha. Además hay que tener cuidado a la hora de cerrar y abrir los bucles, para que no se nos olvide ni corchete { }.

4. Para calcular el valor de SumaC, usaremos una estructura muy recurrente en los lenguajes de programación, los sumatorios. Para ello, SumaC comienza valiendo 0, y a ese valor le iremos sumando cada valor de v según el bucle i.
5. Para el producto escalar, usaremos también una estructura de sumatorio, con la única diferencia de multiplicar componente a componente los vectores v y w.
6. Cuando se nos pida construir una tabla, procederemos a hacer lo mismo que haríamos con una matriz, construimos unos vectores que almacenen la información que queremos que haya en cada fila de la tabla. Una vez hecho esto, se introducen en el comando data.frame, según el orden que se pida.
   NOTA: También se podría pensar que esta tabla se podría hacer con una matriz, ya que el procedimiento es muy similar, pero en este caso los valores de la matriz seguirían siendo tratados como nombres, mientras que en una tabla se pueden usar posteriormente como valores y trabajar con ellos.

SOLUCIÓN

```
#1.Suma de dos vectores
v=c(12,-3,5,18.7)
w=c(12,0.25,77,exp(2))
#n toma el valor de la longitud de v
n=length(v)
#se inicializa z como z=c(0); o como z=o
z=c(0)
#bucle para i desde 1 hasta n =(i in  1:n)
for(i in 1:n){
        z[i]=v[i]+w[i]
}
z
#Comprobación de la suma
v+w

#2.Suma componentes de v y almacenarlo en SumaC
SumaC=v[1]+v[2]+v[3]+v[4]
SumaC=0
for(i in 1:n){
        SumaC=SumaC+v[i]
}
SumaC
#3.Producto escalar de v y w y comprobarlo usando %*%
pesc=0
for(i in 1:n){
        pesc=pesc+v[i]*w[i]
}
pesc
v%*%w   #es lo mismo que hacer el producto escalar sin usar todos los bucles anteriores

#4.Multiplicar ambos vectores componente a componente
v*w    #Producto de v y w componente a componente.

#EJEMPLO: Se tiene un vector rho que contiene valores de la densidad de una sustancia
```

```
rho=c(10,28,0.23,14.4)
#y otro vector vol que contiene los volúmenes de la sustancia
vol=c(90,23,56.5,48)
#y se quiere calcular la masa
masa=0
masa=rho*vol
masa
#También se podría hacer con bucles
Masa=0
for(i in 1:length(rho)){
        Masa[i]=rho[i]*vol[i]
}
Masa

#5.Realizar la operación...
z=0
for(i in 1:n){
        z[i]=v[i]+2*w[i]
}
z

#6.Contruir la tabla...
nombres=c("Suma","Producto escalar")
resultados=c(SumaC,pesc)
data.frame(nombres,resultados)
```

2.- Cálculo del IMC medio y representación de valores aleatorios de IMCs

*[Es recomendable haber visto o entender el funcionamiento de los condicionales antes de hacer este ejercicio, para poder afrontar el Apartado 10 del ejercicio]

PASOS A SEGUIR

1. Define un vector, llamado Nombres, con los nombres de 5 amigos o familiares.

2. Define un vector, llamado Estatura, con la estatura, en cm, de esas 5 personas.

3. Define un vector, llamado Peso, con el peso en kg de esas 5 personas.

4. Calcula el Índice de Masa Corporal = Peso/(Estatura^2) y almacenarlo en el vector IMC.

5. Construye un data.frame que contenga los nombres, estaturas, peso e IMC de cada persona.

6. Genera un vector llamado pesos, que contenga una distribución aleatoria de pesos de 100 individuos en el intervalo [60,90]. Usa runif(100,60,90).

7. Genera un vector llamado estaturas, que contenga una distribución aleatoria de estatura de 100 individuos en el intervalo [1.60,1.90].

8. Obtén un vector llamado imcs que contenga el índice de masa corporal de cada individuo.

9. Realiza una representación gráfica de los imcs de cada individuo tipo histograma (empleando 'h').

10. Calcula el imc medio de la población, señalando cual es su situación según la tabla:

Índice de Masa Corporal	Tu rango
15 o menos	Delgadez muy severa
15 – 15.9	Delgadez severa
16 – 18.4	Delgadez
18.5 – 24.9	Peso Saludable
25 – 29.9	Sobrepeso
30 – 34.9	Obesidad Moderada
35 – 39.9	Obesidad severa
40 o más	Obesidad muy severa (obesidad mórbida)

RAZONAMIENTO

1. Los primeros pasos, en los que se tiene que generar los vectores y sus posteriores operaciones, ya los hemos visto en anteriores ejercicios. Lo que podría surgir como algo nuevo es la generación de los pesos aleatorios. Para ello, usaremos el comando runif(), en el que estableceremos la cantidad de datos que queremos crear en la primera variable, y en las dos siguientes, los límites entre los que aparecerán los valores. En el vector de alturas aleatorias, se procedería de igual forma.
2. Una vez obtenido los vectores “pesos” y “alturas”, nos resultará sencillo operarlos, ya que con una operación básica, cada componente de estos dos vectores se operará y quedarán expresadas en un nuevo vector “imcs”, el cual tendrá una misma longitud que nuestros vectores de partida.
3. Para la representación gráfica de los imcs, hay que tener en cuenta, que en esta ocasión nuestros datos de entrada de la función son dos: pesos y alturas. De esta manera obtendremos la longitud que tendrán las líneas verticales de nuestra función.
4. Para que quede bien estructurado, a la hora de hacer la representación gráfica, podemos crear un vector que vaya desde un valor 1 hasta el 100, con intervalos de una unidad (x=c(1:100)). Este nuevo vector lo enfrentaremos a nuestros valores previamente calculados con la función imcs; y para que quede con un estilo de histograma (barras), escribiremos dentro del comando plot() el type=”h”.
   NOTA: La gráfica nos debería quedar como diez barras verticales con diferentes alturas, aunque con una mayor tendencia a estar por el centro de los valores del EJE Y.
5. Para el último apartado, usaremos varias estructuras de condicionales seguidas, para poder relacionar el valor medio de imcs que hayamos obtenido con su correspondencia en la tabla que se nos proporciona.
   - Comenzamos calculando el valor medio de los imcs. Para una mayor comodidad, llamaremos al valor de la suma de todos los imcs como suma (suma=sum(imcs)), y lo dividiremos entre 100 (número de elementos). A este valor lo podemos llamar “y”, para que nos sea más rápido escribirlo todas las veces que sea necesario en los condicionales.

- En el primer condicional, establecemos que si y tiene un valor menor que 15, nos contestará “Delgadez muy severa”.
- En el siguiente condicional, al escribir else if, le estamos diciendo al programa que si y es mayor a el valor previamente asignado y si es menor al nuevo valor, expresará una nueva respuesta. Esto lo seguiremos haciendo con el resto de condicionales, hasta terminar todos los condicionales necesarios.

NOTA: Debemos entender que la condición else if, pasa a tomar parte cuando la condición anterior no se ha cumplido. Es por ello que no ponemos que debe ser mayor que el valor anterior. Pero cabe destacar que si se pusiera esta doble condición con el nexo &, también estaría bien el procedimiento, pero añadimos información que el ordenador va a ignorar.

NOTA: Una buena manera de intuir que lo hemos hecho bien, es que la gran mayoría de las veces nos de como respuesta “Peso saludable”, ya que por probabilidad, la media de imcs debe oscilar entre su intervalo.

SOLUCIÓN

```
#1.Vector Nombres
Nombres=c("Juanma","Mar","Angélica","Jorge","William")
#2.Vector Estatura
Estatura=c(1.85,1.65,1.70,1.80,1.70)
#3.Vector Peso
Peso=c(80,60,55,85,75)
#4.Cálcula del IMC
IMC=Peso/Estatura^2
#5.Tabla
data.frame=c(Nombres,Estatura,Peso,IMC)
#6.Vector aleatorio pesos
pesos=runif(100,min=60,max=90)
#7.Vector aleatorio estaturas
estaturas=runif(100,min=1.60,max=1.90)
estaturas2=estaturas^2
#8.Vector aleatorio imcs
imcs=pesos/estaturas^2
#9.Representación gráfica de los imcs
imc=function(pesos,estaturas){
        pesos/estaturas2
}
x=c(1:100)
plot(x,imc(pesos,estaturas2),type="h")
#10.imc medio de la población
suma=sum(imcs)
IMCmedio=suma/100
y=IMCmedio
if(y<15){
        "Delgadez muy severa"
} else if(y<15.9){
        "Delgadez severa"
} else if(y<18.4){
        "Delgadez"
} else if(y<24.9){
        "Peso saludable"
```

```
} else if(y<29.9){
        "Sobrepeso"
} else if(y<34.9){
        "Obesidad moderada"
} else if(y<39.9){
        "Obesidad severa"
} else if(y>40){
        "Obesidad muy severa"
}
```

PRÁCTICA 4:

1.- Sistema electoral d´Hondt aplicado a los resultados en las selecciones de España

*[Se enfrentarán los resultados reales (52 circunscripciones) con nuestro resultado, orientado a un uso de una única circunscripción (España es interpretada como una única provincia)]

PASOS A SEGUIR

1. PROGRAMAR:
asignacion_escaños<-function(votos,npart,nesc) Y SALIDA: escaños_unica con los siguientes pasos:

- Inicializar los contadores: k=1;cuenta_escan=1
- Inicializar a 0 una matriz A de npart filas y nesc columnas.
- Inicializar el vector escaños_unica = 0. Además, hacer: escaños_unica[i]=0, (i=1,…npart)
- Mientras (cuenta_escan<nesc) hacer:
  - Almacenar en la columna k de A la cantidad: votos[i]/k (i=1,…,npart)
  - Encontrar el mayor valor de la matriz A (filas 1:npart ; columnas 1:k) almacenando el valor en la variable max, su fila en imax y su columna en jmax.
  - Hacer A(imax,jmax)=0 para no volverlo a utilizar.
  - Hacer escaños_unica [imax]=escaños_unica [imax]+1; k=k+1;cuenta_escan=cuenta_escan+1

Fin del bucle
return (escaños_unica)
Fin function

2. DATOS:
nesc=350; npart=17
Votos=c(6752983, 5019869, 3640063, 3097185, 1637540, 869934, 577055, 527375, 377423, 276519, 244754, 226469, 123981, 119597, 98448, 68580, 19696)
Partido=c('PSOE','PP','VOX','PODEMOS','Cs','ERC','MasPais','JxCAT','PNV','BILDU','CUP', 'PACMA', 'CC', 'BNG', 'NAVARRA_SUMA','PRC','TeruelExiste')
escaños_reales=c(120, 89, 52, 35, 10, 13, 8, 8, 6, 5, 2, 0, 2, 1, 2, 1, 1)

RAZONAMIENTO

1. Para empezar, iniciaremos la cuenta en 1, que iremos aumentando conforme se vayan completando cada bucle con sus respectivos valores. Para evitar errores y hacer la tarea más sencilla, iniciamos la matriz A a 0.
2. Con cada división sucesiva en la operación votos[i]/k, lo que conseguimos es ir dividiendo por 2 la matriz A, para luego encontrar el valor más alto contando con las columnas anteriores y las que se hayan agregado al dividir.
   NOTA: Hay que tener en cuenta que en ningún momento el número de filas va a variar.
3. Según el programa va seleccionando los datos más altos, estos datos se van almacenando en la función asignación_escaños. Estos valores se pasarán después a nuestro vector escaños_unica.
4. Para poder contrastar los resultados obtenidos frente a los reales y que sea más fácil poder compararlos, los ponemos en una misma tabla mediante el comando data.frame.

SOLUCIÓN

*[No es necesario aplicar todas las sangrías, pero es recomendable para mantener un mejor control de dónde empieza y acaba cada bucle]

```
asignacion_escaños=function(votos,npart,nesc){
        k=1 ; cuenta_escan=1
        A=matrix(c(0),nrow=npart,ncol=nesc)
        escaños_unica=0
        for(i in 1:npart){
        escaños_unica[i]=0
        }
        while(cuenta_escan<nesc){
                for(i in 1:npart){
                        A[i,k]=votos[i]/k
                }
                max=0 ; imax=1 ; jmax=1
                for(i in 1:npart){
                        for(j in 1:k){
                                if(A[i,j]>max){
                        max=A[i,j] ; imax=i ; jmax=j
                                }
                        }
                }
                A[imax,jmax]=0
                escaños_unica [imax]=escaños_unica [imax]+1 ; k=k+1 ;
cuenta_escan=cuenta_escan+1
        }
        return(escaños_unica)
}
nesc=350 ; npart=17
votos=c(6752983, 5019869, 3640063, 3097185, 1637540, 869934, 577055, 527375, 377423,
276519, 244754, 226469, 123981, 119597, 98448, 68580, 19696)
Partido=c('PSOE','PP','VOX','PODEMOS','Cs','ERC','MasPais','JxCAT','PNV','BILDU','CUP',
'PACMA', 'CC', 'BNG', 'NAVARRA_SUMA','PRC','TeruelExiste')
escaños_reales=c(120, 89, 52, 35, 10, 13, 8, 8, 6, 5, 2, 0, 2, 1, 2, 1, 1)
escaños_unica=asignacion_escaños(votos,npart,nesc)
T = data.frame(Partido,escaños_unica,escaños_reales)
T
```

PRÁCTICA 6:

1.- Interpolación de Lagrange aplicado a la producción de bioetanol

PASOS A SEGUIR

Estimar, mediante interpolación de Lagrange, la concentración de bioetanol en los instantes t = 2 y t = 6.5. Para ello, se empleará la siguiente expresión del polinomio interpolador de Lagrange.

$$p = \sum_{i=1}^{n} B_i L_i$$

1. Programar la función Polbase para obtener las funciones de base la interpolación de Lagrange, que están dadas por la expresión

$$L_i = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{t - s_j}{s_i - s_j}, (i = 1, \ldots, n)$$

2. Programar la función PolInterp para obtener el polinomio interpolador en el punto x. Dicha función recibirá como argumentos de entrada:
   - un vector B (que contiene los valores de la función que se interpola)
   - el vector L (que contiene los polinomios de base de Lagrange evaluados en el punto t)
   - la variable n (número de puntos del soporte de interpolación.
3. Introducir los datos y ejecutar las funciones programadas. Al ejecutar, se almacenarán las funciones de base en el vector L y el polinomio interpolador en la variable p.
4. Obtener 1001 abscisas equidistantes en el intervalo [s[1],s[n]] y almacenarlas en un vector x. (Calcularemos el valor del polinomio interpolador en cada abscisa para dibujar).
5. Llamamos 1001 veces a las funciones Polbase y PolInterp. La salida de Polinterp se guardará en un vector f[k], k=1,…,1001.

DATOS:

Los valores de la concentración (g/l) están almacenados en el vector: B(10,20,35.33,35.5) y los instantes de tiempo (horas) en el vector s(1,3.5,5,10).

RAZONAMIENTO

1. Creamos la función Polbase antes de introducir las variables ya que resulta más sencillo introducirlos más tarde.

   NOTA: Hacemos el bucle correspondiente como cualquier otro, pero tenemos que tener en cuenta que j no puede ser igual a la i. Para ello, usamos el comando i!=j que indica la desigualdad. Es importante tener presente que para poner esta condición a la j, se debe dar el valor de i antes.
2. Una vez calculado la función Polbase, podemos pasar a definir la función Polinterp (depende del valor de Polbase).
3. Se introducen los datos y escribimos n como la longitud del vector B, aunque también podría haberse cogido el vector s. se calcula para t=2 y después para t=6.5.

NOTA: Se pueden dar otros nombres a las variables, pero hay que tener cuidado con poner las variables siempre en el mismo orden, que es lo que tendrá en cuenta el programa.

4. Antes de definir el vector x, definimos cual debe ser el intervalo que deben tener los valores del vector.

   NOTA: La longitud del vector x, también se puede expresar como length=1001 en vez del valor que hayamos calculado en h.

5. Al dibujar las funciones, veremos que no coinciden. Esto es porque tienen una diferente escala. Para poder solucionarlo, escribimos unos límites (xlab, ylab) iguales en cada función, y podemos darles diferentes colores para distinguirlas mejor.

   NOTA: Cuando escribimos f[k]=p, podemos eliminar esto si antes igualamos el Polinterp(n,L,B) a f[k] directamente.

SOLUCIÓN

```
Polbase=function(n,t,s){
L=0
for(i in 1:n){
        L[i]=1
        for(j in 1:n){
                if(i!=j){
                L[i]=L[i]*(t-s[j])/(s[i]-s[j])
                }
        }
}
return(L)
}
Polinterp=function(n,B,L){
p=0
for(i in 1:n){
        p=p+B[i]*L[i]
}
return(p)
}
#Introducción de datos
s=c(1,3.5,5,10)
B=c(10,20,35.33,35.5)
n=length(s)
t=2
L=Polbase(n,t,s)
L
p=Polinterp(n,L,B)
p
t=6.5
L=Polbase(n,t,s)
L
p=Polinterp(n,L,B)
p
h=(s[n]-s[1])/1000
x=seq(s[1],s[n],h)
f=0
```

```
for(k in 1:1001) {
        t=x[k]
        L=Polbase(n,t,s)
        p=Polinterp(n,L,B)
        f[k]=p
}
f
plot(s,B,pch=19,xlab="x",ylab="Concentración(g/L)",main="Produccióndе
bioetanol",xlim=c(s[1],s[n]),ylim=c(0,60))
par(new=TRUE)
plot(x,f,col="blue",xlim=c(s[1],s[n]),ylim=c(0,60),type="l",xlab=" ",ylab=" ")
```

2.- Métodos de integración numérica: "Método de Simpson"

PASOS A SEGUIR

1. Se considera una sustancia cuya densidad viene dada por d(x) siendo x la coordenada espacial. La masa total en cierto intervalo [A,B] está dada por:

$$M_{AB} = \int_A^B d(x)dx$$

2. Se desea realizar un programa en R para resolver dicha integral mediante una fórmula de Simpson compuesta, cuya expresión viene dada por:

$$\int_A^B d(x)dx \approx \frac{h}{6}\left(d(A) + 2\sum_{i=2}^{n-1} d(T_i) + 4\sum_{i=1}^{n-1} d\left(\frac{T_i + T_{i+1}}{2}\right) + d(B)\right)$$

3. Donde se ha realizado una subdivisión del intervalo [A,B] en (n-1) subintervalos iguales, es decir, considerando n puntos Ti , y siendo h la longitud de cada subintervalo.
4. El programa se ejecutará con los siguientes datos: A=0, B=3.05, densidad dada por d(x)=exp(x)sin(x). Como número de puntos para el desarrollo de la fórmula se tomará n=35.

RAZONAMIENTO

1. Para calcular la integral de la Masa Total, usamos una aproximación de la fórmula de Simpson, donde tenemos que definir antes el valor h y los dos sumatorios de la ecuación.
2. Para el primer sumatorio (S1) usamos un while y para el segundo (S2), usamos un bucle for, para demostrar que en muchas ocasiones tienen una misma funcionalidad.
   NOTA: Es importante poner en el while i=i+1, porque sino, no aumentaría el valor de la i y el while se quedaría siempre en un mismo valor, por lo que daría siempre el mismo resultado.
3. Para ver que el resultado de la aproximación es muy acertado, usaremos el comando integrate(), con los valores d, A y B; de esta manera podremos ver como la aproximación de Simpson (con estos decimales) da el mismo valor que nos daría una operación matemática al calcular la integral entre los puntos A y B.

SOLUCIÓN

```
d=function(x){
exp(x)*sin(x)
}
A=0 ; B=3.05 ; n=35
h=(B-A)/(n-1)
T=seq(A,B,h)
S1=0
i=2
while(i<=(n-1)){
        S1=S1+d(T[i])
        i=i+1
}
S1
S2=0
for(i in 1:(n-1)){
        S2=S2+d((T[i]+T[i+1])/2)
}
S2
M=h*(d(A)+2*S1+4*S2+d(B))/6
Print("Con Simpson")
M
integrate(d,A,B)
```